PC-8801/mkII/SR

解析マニュアル 総集編

増渕文人・茜照夫・島崎孝

SHUWA
SYSTEM
TRADING
CO.,LTD.

PC-8801/mkII/SR
解析マニュアル 総集編

増淵文人　　茜　照夫　　島崎　孝

SHUWA SYSTEM TRADING CO., LTD

はじめに

発表と同時に大きな注目を浴びたＰＣ－８８００シリーズも、はや５年目を迎えようとしております。しかし、その人気は衰えを知らず、モデルチェンジごとにますます多くのユーザの支持を集めて今日に至っています。弊社では、この優れた機能を最大限に引き出すための技術資料として解析シリーズ全４巻と、これらの活用例としてユーティリティプログラム集を刊行し、幸いにも皆様の高い評価を得ることができました。これらの利用により、マシン語応用プログラムにおいてＲＯＭ内ルーチンを使用することなどが、極めて容易になったことと思います。

このように、私たちは内部ルーチンを自由自在に使いこなせるようになりました。しかしながら、ここで新たな疑念が生じてきます。自由に操れるとはいっても、その内部については何もわかっていないではないか。これでは、コンピュータの構造など何も知らない子供がファミリーコンピュータで遊ぶのと同じではないか……。このような皆様の疑念に答えるべくして今回発表するのが、この「総集編」というわけなのです。ＦＯＲ文やＷＨＩＬＥ文などの制御構造の実現方法、あるいはグラフィック画面に円や直線を描くアルゴリズム、あるいは三角関数を求めるルーチンの作り方、そして何よりも、ＢＡＳＩＣインタプリタの大局的な把握の仕方など、この本ではプログラムの動作原理について深く追究していきます。いままで何気なく使っていたコマンドも、その背景にはさまざまな技術の蓄積があることにきっと驚かれることと思います。また、従来にも増して多くの図を採り入れたことは、必ずや皆様の直観的な理解の助けとなることでしょう。

内部のメカニズムについて知識がほとんどなくても、なんとか車は動いてくれます。しかし、運転技術にははっきりその差が現れるともいいます。本書が皆様のプログラム作成技術に少しでも寄与できるのであれば、これ以上の喜びはありません。

最後になりましたが、この本の完成までには菊地寿、斉藤和邦、立野剛志の各氏から惜しみない助力をいただいたことを一言記させていただきます。

１９８５年　１２月　　　　　　著者一同

参　考　文　献

PC-8801mkⅡSR N88-BASIC REFERENCE MANUAL

PC-8801mkⅡSR N88-BASIC USER'S GUIDE　　以上　日本電気

PC-Techknow 8801 Vol.1　　株式会社アスキー

PC-Techknow 8801mkⅡ　　株式会社システムソフト

PC-8801 マシン語活用ハンドブック

PC-8801 N88-BASIC 解析マニュアル

PC-8801 N88-BASIC ディスク解析マニュアル

PC-8801 N88-BASIC グラフィクス解析マニュアル

PC-8801 N88-BASIC モニタ解析マニュアル

PC-8801mkⅡSR テクニカルマップⅠ ソフトウェアを知る

PC-8801mkⅡSR テクニカルマップⅡ ハードウェアを知る

PC-80S31 フロッピディスク活用ハンドブック

FM-7 F-BASIC 解析マニュアル フェーズⅡ(探究編)

以上　秀和システムトレーディング株式会社

目　次

1

第1章

PC-8801の構成とN88-BASIC

CONTENTS

１－１　メモリ空間の構成

ＰＣ－８８０１のＣＰＵにはＺ８０コンパチブルのμＰＤ７８０が使われています。このＣＰＵが直接アクセスすることのできるメモリは最大64Ｋバイト（65536バイト）です。ところが、N_{88}－ＢＡＳＩＣの大きさは54Ｋバイトで、システムワークエリアも含めると、プログラムテキストや変数エリア等に使用できる領域はほとんどありません。ＰＣ－８８０１では、すべての領域を同時にアクセスする必要がないことに目をつけ、メモリのセレクト信号を制御することによってとりあえず不要なところはＣＰＵのメモリ空間から切り離し、適宜に必要な部分のみを接続していきます(これをバンク切り換えという)。このようにして、ＣＰＵの240Ｋバイトのメモリアクセスを可能にしています。

ＰＣ－８８０１のメモリ構成を図１－１－１に示します（ＳＲについては後述)。

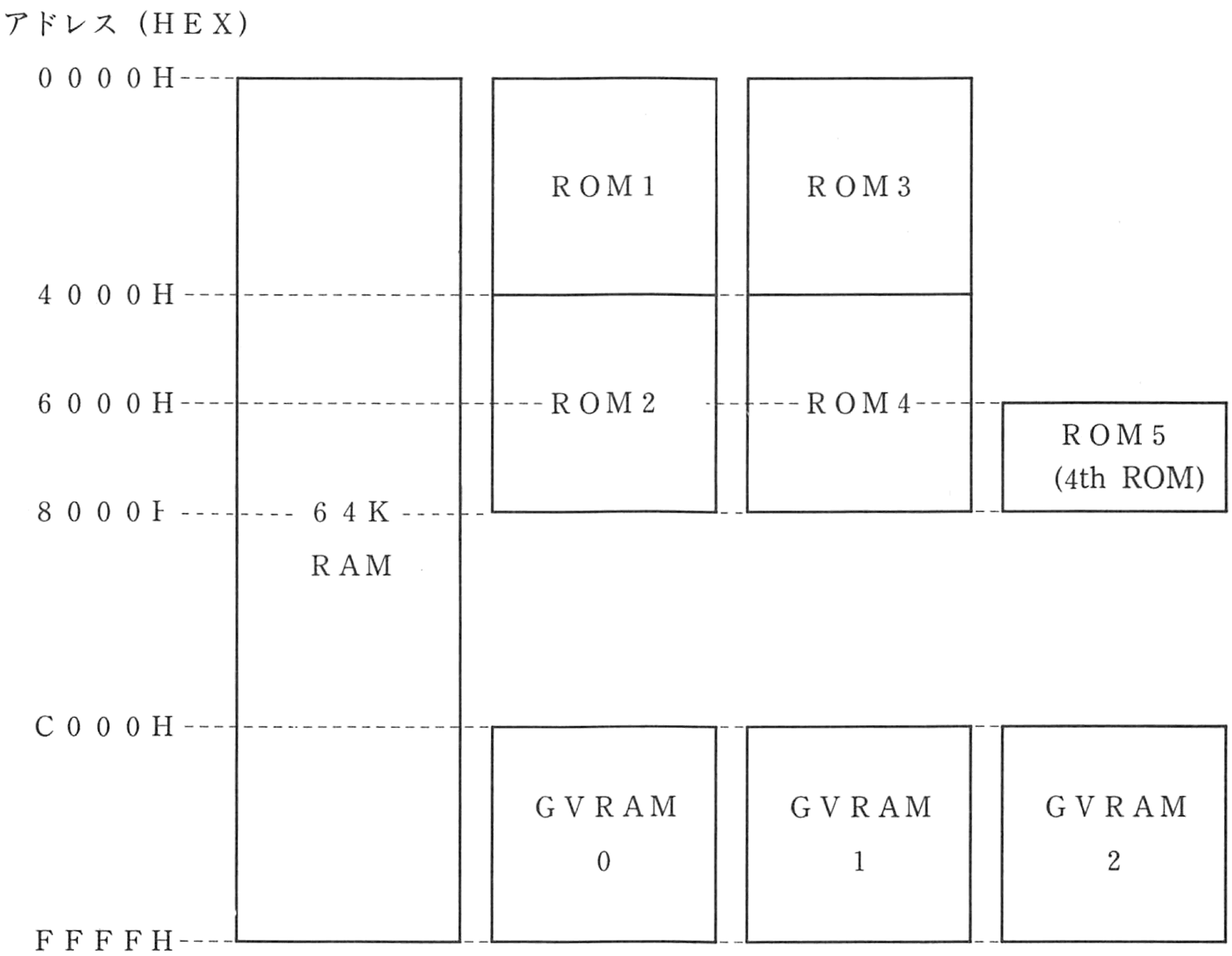


図１－１－１　メモリバンク（標準状態）

この図は、ＰＣ－８８０１／mkⅡに標準で実装され、かつＣＰＵのアドレス空間に出現するメモリの種類とアドレスです。このうち、ＲＯＭ１とＲＯＭ２（Ｎ－ＢＡＳＩＣモード）は同時に現れ、ＲＯＭ５はＲＯＭ３・４セレクト時のみ有効となります。ＲＯＭ１・３は００００Ｈ～３ＦＦＦＨ、ＲＯＭ２・４は４０００Ｈ～７ＦＦＦＨ、ＲＯＭ５は６０００Ｈ～７ＦＦＦＨに位置しています。また、ＧＶＲＡＭ０～２は高分解能グラフィクス用ＶＲＡＭで、すべてＣ０００Ｈ～ＦＦＦＦＨに位置しています。メインＲＡＭは64Ｋバイトあり、００００Ｈ～ＦＦＦＦＨのすべてに渡っています。このうち、アドレス００００Ｈ～７ＦＦＦＨについては、ＲＯＭと重なりますが、ＲＯＭの状態にかかわらず常に書き込み可能になってはいます。

また、アドレス８０００Ｈ～８３ＦＦＨは特殊な構造になっています。テキストウィンドウと呼ばれるこの領域は、オフセットレジスタ（入出力ポート７１Ｈによってアクセスできる）を設定することによってメインＲＡＭの任意のアドレスをアクセスすることができます。オフセットレジスタの値とこの領域に出現するメインＲＡＭ上のアドレスの関係は、ウィンドウ内アドレスをＸ、オフセットレジスタ値をＣ、メインＲＡＭアドレスをＩとすると、

$$I = C \times 100H + X - 8000H$$

となります。たとえば、Ｃ＝２１Ｈ、Ｘ＝８２５１Ｈとすると、アクセスされるメインＲＡＭアドレスＩ＝２３５１Ｈです。ちなみに、ウィンドウ領域の裏側のＲＡＭも同様にアクセスでき、Ｃ＝８０Ｈとすればウィンドウアドレスと実アドレスが一致します。オフセットレジスタの操作には出力ポート７８Ｈも用いられ、このポートに任意の値を出力するとレジスタの内容がインクリメントされます。テキストウィンドウは、N_{88}－ＢＡＳＩＣでＲＯＭの裏側をＢＡＳＩＣテキスト格納に用いるために作られたもので、００００Ｈ～７ＦＦＦＨにＲＯＭがセレクトされているときでも、この領域には前述のアドレスのメインＲＡＭの内容が現れています。

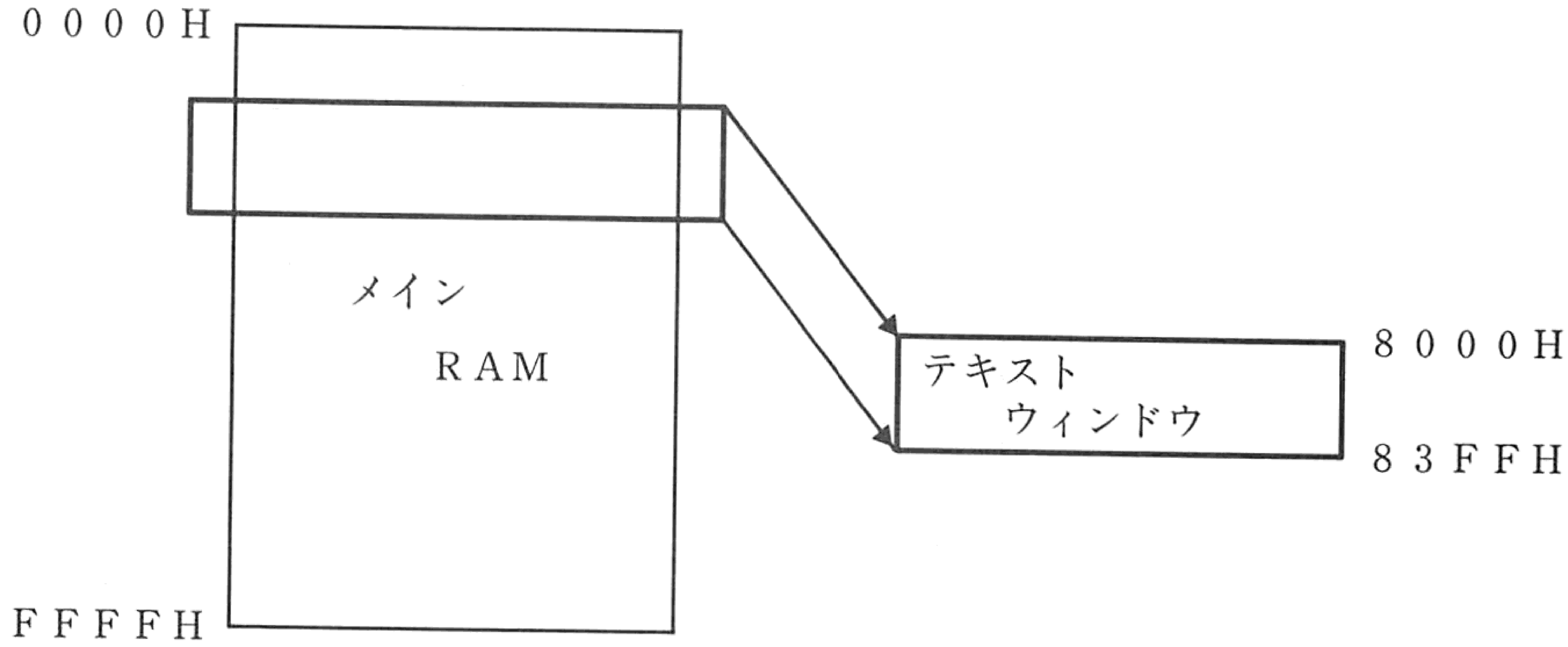


図１－１－２　テキストウィンドウ

ここで注意しなければならないのは、N－BASICモード（ROM1・2がセレクトされているとき）やオールRAMモード（0000H～7FFFHの領域でメインRAMがセレクトされているとき）では他機種と互換性をもたせる関係上、ウィンドウ機能は働かないということです。

以上、標準状態のメモリ配置について述べましたが、これらの他に、拡張ROM、拡張RAMがあります。

拡張ROM（4thROMと呼ばれる）は、ROM5と同じアドレスに位置し、最大7バンクまで拡張することができます。実はROM5とは4thROMの第0バンクにほかならず、セレクト方法も拡張ROMとROM5は同様な方法をとっています。すなわち、ポート71Hの第nビットを0にすることで第nバンクをセレクトします。もちろん、拡張ROMがセレクトされるのはN88－BASICモードのときに限ります。拡張ROMが現れているとき、それがメモリのアクセスに影響するのは6000H～7FFFHだけです。

拡張RAMは、0000H～7FFFHに位置します。このRAMに対する読み書きはメインRAMとは異なりバンクセレクトされた状態以外では行えません。このRAMをセレクトするにはポートE2Hを用います。

今までは0000H～7FFFHでセレクトされるメモリ名でモードを示しましたが、これらに名称をつけることにします。それを図1－1－3に示します。

セレクトされているメモリ	名　称（モード）
メインRAM	オールRAM
ROM1，2	N－BASIC
ROM3，4	N88－BASICノーマル
ROM3，4，5	〃　EROM
4thROM　n	〃　4thROM　n

図1－1－3　バンク切り換えモード

表中、EROMという語がでてきましたが、これは、External ROM（外部ROM）という意味です。

図1－1－4はROMの各状態の名称とその状態を実現するために操作するポート（およびそのビット）です。

MMODE・RMODEはともに出力ポート31Hのビットの名前で、それぞれ第1ビット・第2ビットとなっています。4thROMは入出力ポート71Hです。これらのポートにカッコ内の値を出力することによって、それぞれの枠内のモードを得ることができます。たとえば、MMODE＝0，RMODE＝0，4thROM＝FEHならばN88－BASICのEROMモード

となるわけです。また、斜線の部分はその下のポートに無関係に左のモードとなることを示しています。

<table>
<tr><td>ポート</td><td>31Hの
ビット0</td><td>31Hの
ビット1</td><td>ポート71H</td></tr>
<tr><td rowspan="5">モード（カッコ内は各ビットの値）</td><td>オール
ROM
モード
(1)</td><td colspan="2"></td></tr>
<tr><td rowspan="4">RAM
共　存
モード
(0)</td><td>N－BASIC
(1)</td><td></td></tr>
<tr><td rowspan="3">N_{88}－BASIC
(0)</td><td>ノーマル（FFH）</td></tr>
<tr><td>EROM（FEH）</td></tr>
<tr><td>4thROMn
(第nビットを0)</td></tr>
<tr><td></td><td>MMODE</td><td>RMODE</td><td>4thROMn</td></tr>
</table>

図1－1－4　ROMモード

逆に、どのポート（のビット）がどのROMをセレクトするかを表にしたのが図1－1－5です。

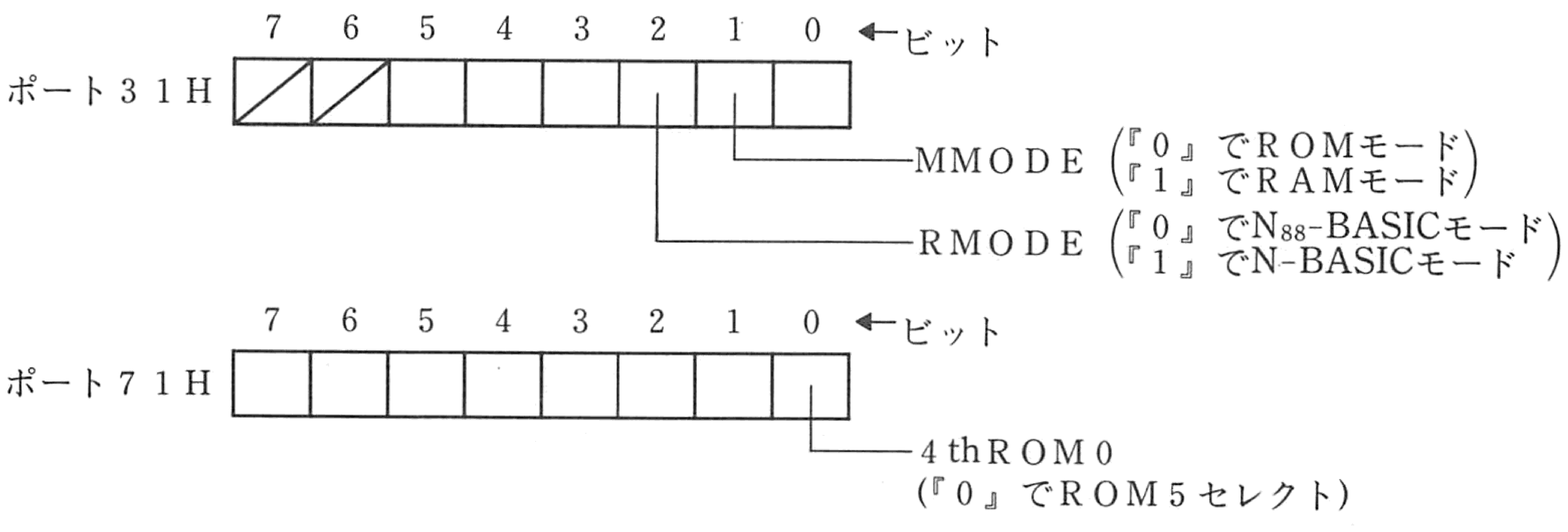


図1－1－5　メモリセレクト

１―２　ＢＡＳＩＣの起動

ＰＣ－８８０１の電源スイッチを入れてからＢＡＳＩＣのコマンド入力モードになるまでには、さまざまな初期化ルーチンを通らなければなりません。これらは、ハードウェア初期化とＢＡＳＩＣ作業域初期化に分けることができます。

ハードウェア初期化には、Ｉ／Ｏ初期化ルーチン（７２ＣＤＨ）が用いられています。このサブルーチンの詳細は入出力の章を参照していただきたいのですが、概略を述べると、①入出力関係ポート、②漢字ＲＯＭ、③割り込みコントローラ、④ディスクおよび関係作業域、⑤画面表示、の初期化を行います。

一方、ＢＡＳＩＣ作業域初期化は、すべての作業領域へ、ＲＯＭ上に用意されている初期値を転送するほか、①拡張フック、②ＢＡＳＩＣテキスト関連ポインタ、③ファイルバッファ、④ドライブテーブルの初期化、を行います。

このほか、ディップＳＷによるモード分岐処理、ＳＴＯＰ－ＲＥＳＥＴ時のホットスタート処理、拡張ＲＯＭ上（４thＲＯＭ上）の初期化ルーチンの呼び出しなどが行われます。

ただし、ディスクが接続されているときには、ＲＯＭ上の初期化ルーチンを完遂せずに、ディップＳＷステータス設定とＢＡＳＩＣ作業域初期化だけを行ってＩＰＬルーチンをコールします。通常、ＩＰＬでは独自の初期化ルーチンを実行した後にプログラム本体にジャンプし、ＲＯＭ上の初期化ルーチンに復帰することはありません。

ＮＥＣのＤＩＳＫ－ＢＡＳＩＣも例外ではなく、ＩＰＬから後はすべて、ディスクコード内の初期化ルーチンを用います。このルーチンは、オートスタート機能とドライブテーブル作成を除けば、ＲＯＭ上にあるルーチンとまったく同じ構造を持っています。

１―２―１　ＲＯＭ　ＢＡＳＩＣの起動

ＰＣ－８８０１のシステムにリセットがかかり、ディスクユニットが接続されていないか、またはディスクが入っていないときにはＲＯＭ－ＢＡＳＩＣが起動されます。もちろん、このときにはＲＯＭ上の初期化ルーチンが用いられます。

また、ＤＩＳＫ－ＢＡＳＩＣ起動の場合でも、Ｎ－ＢＡＳＩＣモードへの切り換え・分岐と拡張フックの初期化はＲＯＭ上のルーチンが行います。これは、リセットがかかるとN88－ＢＡＳＩＣメインＲＯＭが自動的にセレクトされるためで、ディップＳＷの読み出しとそれに従ったＲＯＭのセレクトはソフトウェアで行わなければなりません。ところが実際の処理は、リセット直後にディップＳＷの読み出しと作業域への設定を行っているので、後に実行されるモード分岐のときに、どちらのＢＡＳＩＣモードの状態でリセットがかかったかを判別することができず、常にディップＳＷの状態だけを見て分岐します。したがって、「ＮＥＷ　ＯＮ命令」で

N−BASICモードにしたときにはSTOP−RESETが使用できないので注意が必要です。

この処理が終わると、N_{88}−BASICコールドスタート処理（7820H）となります。こ

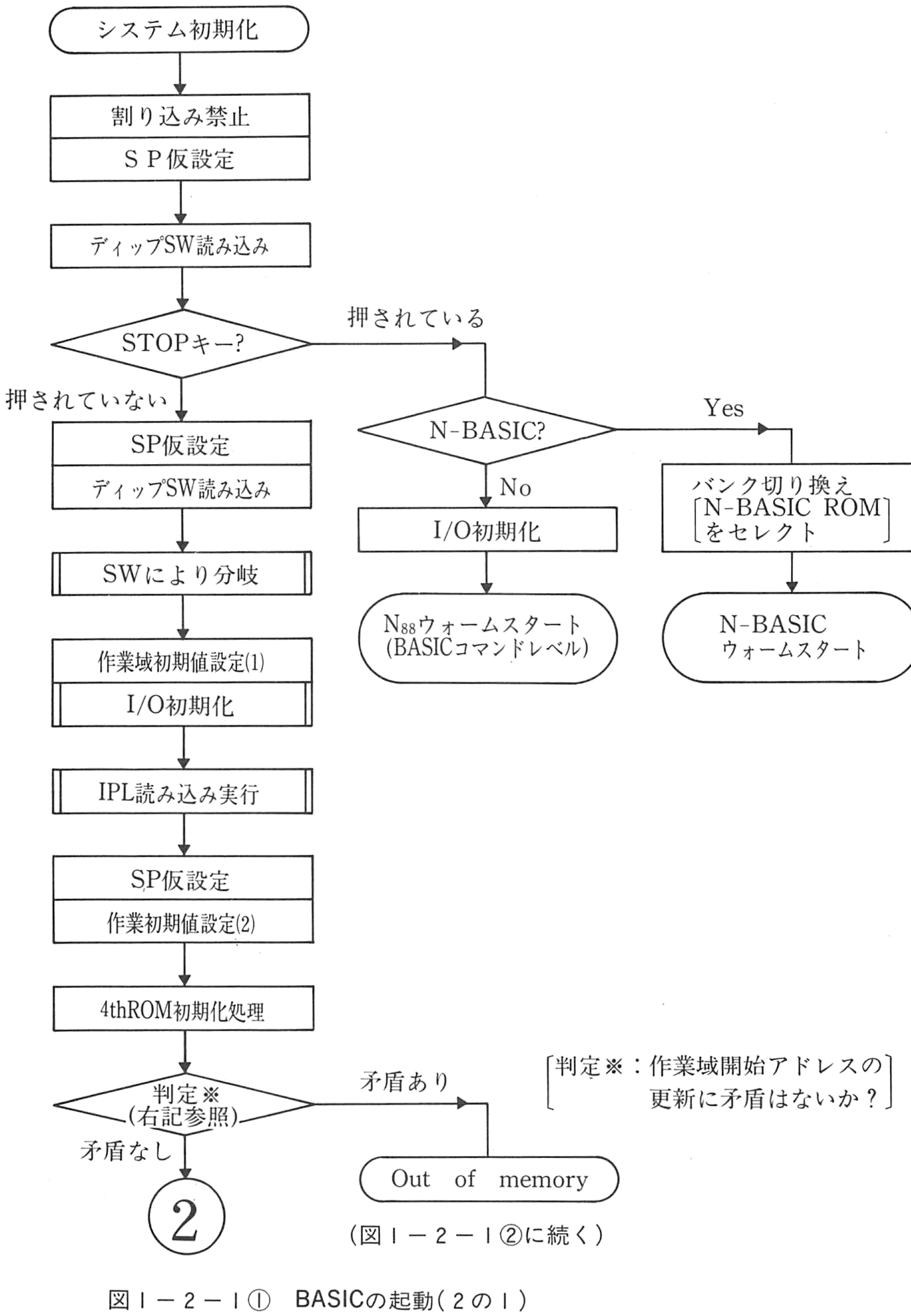


図1−2−1①　BASICの起動（2の1）

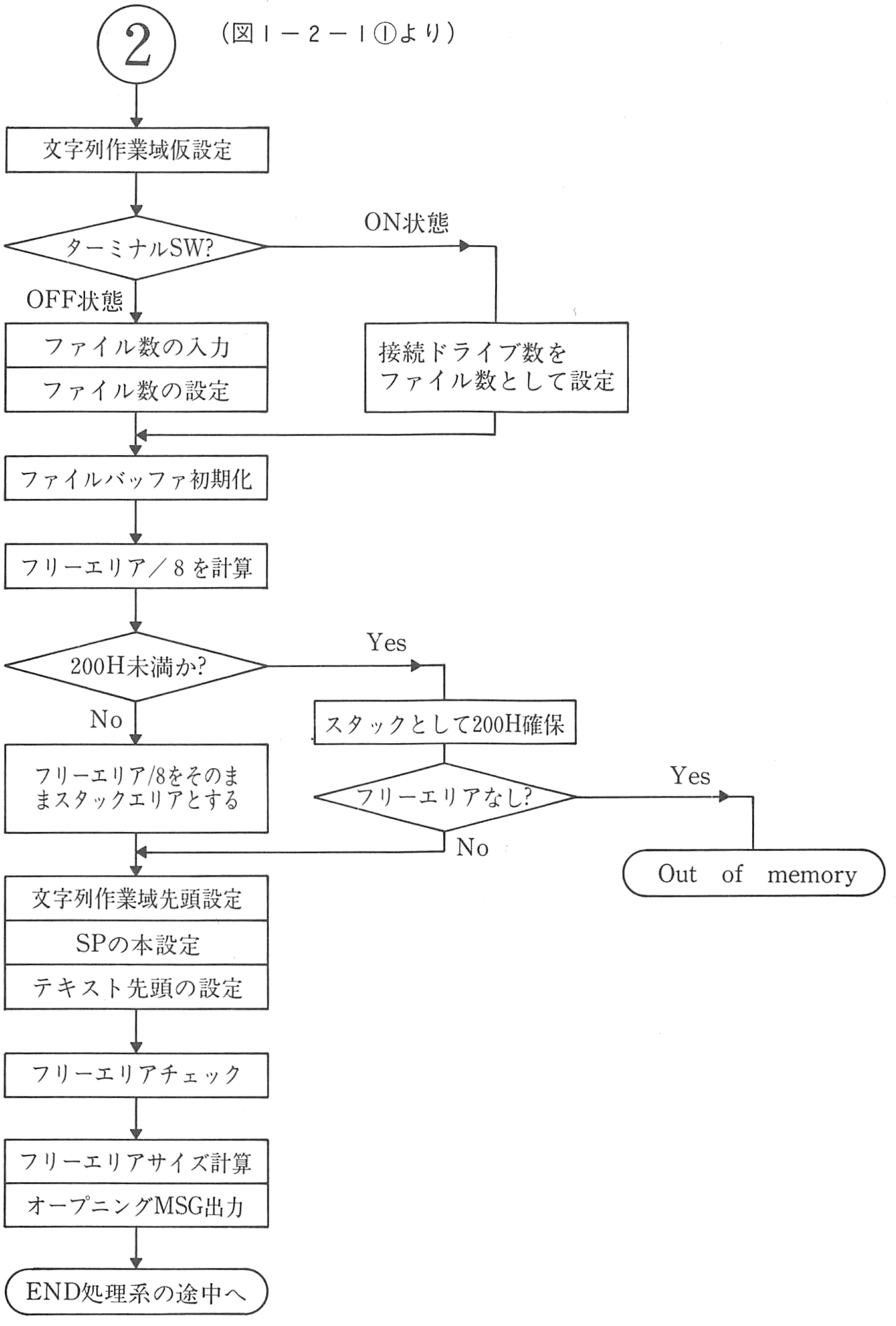


図１－２－１②　BASICの起動（２の２）

のルーチンにエントリするにはDEレジスタにディップSWの１の補数を持っていなければなりません。これは、「NEW　ON命令」でソフト的にディップSWの状態を設定するためです。DレジスタにSW２、EレジスタにSW１が対応しています。

ディップSWステータスに従った実際のROMのセレクト、ジャンプはROM切り換え処理（７７９CH）中で行われます。ただし、このルーチンにはターミナルモードへの移行（ディップSW１のビット１がOFFのとき）も含まれ、初期化のこの段階では作業域設定が十分に行われていないので、ターミナルモードとならないように途中の７７A６Hにエントリしています。

作業域初期値設定①では、作業域のクリアとROM上にある作業域初期値データ（００BEH～０３３１H）を作業域（E６００H～）に転送します。これで、設定の必要なほとんどの作業域に正当な初期値が設定されたことになります。

これにより、I／Oに関する初期値も確定したので、I／Oの初期化（７２CDH）を実行します。

また、その直前には拡張フック初期化（６９B２H）が行われます。これは、拡張フック（ECBBH～）のうち、サブルーチン内でコールされているものにはRET命令を、構文解析ルーチンからジャンプしてくるものに「JP　４DC１H」なる命令を書き込むものです。I／O初期化処理にも拡張フックがあるので、必ず前もって行わなければなりません。普通は、１度実行すれば十分です（プログラムの暴走などによってテーブルが破壊されたときなどは当然必要）。IPL読み込み実行（４５８FH）は、①ドライブ対応表の作成と関連作業域の設定、②IPL読み込み実行、を行います。

この後に行われる作業域初期値設定②では、インタプリタ実行に関する作業域を初期化します。それらは、中間語バッファ・FNテーブル・CTRL－Oフラグ・Chain文関連フラグ・エラーコードフラグなどがあります。このうち、FNテーブルの初期化は、CLEAR命令の後半（４FA９H）を用いて行われます。

拡張ROM上に、ユーザ用の初期化ルーチンを置くことができます。そのルーチンは上述の処理が終わった時点で実行されます。ただし、そのためには４thROM（６０００H～）の先頭の２バイトにアスキー文字列「R４」、次の２バイトに初期化ルーチンのアドレスが設定されている必要があります。４thROMが現れると、ちょうど処理しているプログラムの上にかぶさってしまうため、バンク切り換えとコールは前半（０１４２H）にあるサブルーチンで行われます。この処理の直後に行われるチェックは、拡張ROMの初期化ルーチンが用いる作業域のために設定しなおされた作業域開始アドレス（E７E８H）が整合的かどうか、すなわち新たに設定される作業域が、前のものを含んでいるかどうかをチェックします。ただし、この部分のフラグの判定が正しく行われていないため、実際に作業域を確保することはできません。

オープン可能ファイル数の設定（７８A０H）は、立ち上げ時に「How many files?」に答えて入力した数を設定するものです。ユーザは、常に標準入出力バッファ#０をオープンするこ

とができるので、実際にオープン可能なファイル数は、上で設定した値プラス１となります。このパラメータの設定は、基本的にはユーザの入力によりますが、特に指定せずに入力を省略（リターンのみ入力）した場合には、ディスクユニットの接続状態によって解釈が異なります。つまり、ディスクユニットが接続されているときにはユニット数がオープン可能ファイル数となり、そうでないときには「１」となります。ファイル数の入力には行入力ルーチン（５ＦＣ２Ｈ）を用い、数の読み込みには１文字読み込み処理ルーチン（１０Ｈ）を用いています。このとき、ＳＴＯＰキーが押されたり、16以上の数が入力された場合にはこの処理の先頭に戻ります。

しかし、ターミナルモードのときの処理は上とは異なり、接続されているディスクドライブ数をオープン可能ファイル数とします。

オープン可能ファイル数（ＥＣ７ＥＨ）が確定した後にすることはファイルバッファの初期化（７８Ｄ２Ｈ）です。

ディスクドライブが接続されていないときには、ファイルバッファアドレステーブルの開始アドレス（ＥＣ７ＦＨ）は８４００Ｈとなります。このテーブルの直後にファイルバッファが設定され、ファイルバッファ#０から順に割り当てられていきます。すなわち、８４００Ｈ＋「(オープン可能ファイル数＋１)×２」がバッファ#０の先頭アドレスとなります。ここから、ファイルバッファはバッファ本体（データ領域）として256バイト、ファイルコントロールブロック（ＦＣＢ）として９バイト必要なので、ひとつにつき265（１０９Ｈ）バイト確保するために、ファイルバッファアドレステーブルを設定していきます。以上でファイルバッファは初期化されたので、ＣＬＯＳＥフラグ（ＥＣＡ３Ｈ）と、今後よく用いられるファイルバッファ#０のデータ領域開始アドレス（ＥＣ８３Ｈ）をセットします。また、この後に始まるＢＡＳＩＣのラベル・変数領域の開始アドレスとして、ラベルテーブル開始アドレス（ＥＢ１６Ｈ）へ「ファイルバッファ終了アドレス＋１」をセットします。

初期段階で初期化・設定しなければならない作業域は以上がほとんどで、残りは実行に従ってその占める領域や内容が動的に変化するものが大部分です。したがって、ここでスタック領域を設定したあと、立ち上げ時に表示されるフリーエリアサイズを計算します。

スタック領域は、この時点に残っているメインＲＡＭのフリーエリアの８分の１がそれにあてられます。しかし、そのバイト数が２００Ｈに満たない場合は強制的に２００Ｈバイト確保されます。その上で作業域開始アドレスと比較し、メモリが足りない場合には Out of memory エラーとなります。

その後に文字列エリア開始アドレス（ＥＡＣＣＨ）・機械語領域開始アドレス（Ｅ６５４Ｈ）フリーエリア終了アドレス（ＥＡＦ１Ｈ）を設定し、またテキスト開始アドレス（Ｅ６５８Ｈ）の初期化を行い、「文字列エリア開始アドレス－ラベルエリア開始アドレス－２＋８０００Ｈ（テキストエリア）－４」をもってフリーバイト数とします。そしてメッセージを表示してＥＮＤ処理の途中にジャンプします。

Ｉ―２―２　ＤＩＳＫ　ＢＡＳＩＣの起動

ＩＰＬ読み込みの後、続くセクタ、トラックからディスクコードを読み込みます。このうちＡＦ００Ｈ以降は初期化ルーチンにあたり、内容はＲＯＭ上のものを移し替えたものに少し改造を加え、ＤＩＳＫ－ＢＡＳＩＣの初期化をしています。以下、ＲＯＭと異なる部分について解説していきます。

ＩＰＬからＤＩＳＫ－ＢＡＳＩＣ初期化（ＡＦ００Ｈ）に入り、まず、ＤＩＳＫ－ＢＡＳＩＣ用フックの書き換え（Ｂ１２５Ｈ）を行います（ＡＦ０５Ｈ)。そしてＳＰ仮設定、作業域初期値設定②と行います(ＡＦ０８Ｈ)。ＲＯＭ上での次の処理４thＲＯＭ初期化ですが、ディスクコードではこの処理を後回しにします。文字列作業域終了仮設定（ＡＦ３０Ｈ）の後には「Disk Version」を表示し（ＡＦ４３Ｈ)、ディスクのＩＤセクタを読み込みます。オープン可能ファイル数はその１バイト目とみなされます。ただし、ＦＦＨのときは通常のようにキー入力となります（ＡＦ４ＥＨ)。

ＤＩＳＫ－ＢＡＳＩＣではファイルバッファの前にドライブテーブルが設定されます（ＡＦ８２Ｈ)。これは構造としてはファイルバッファと同様で、ドライブポインタとデータ部から成り、データ部にＡ６Ｈバイトとり、ＦＡＴ関係のフラグ・バッファがその内容です。その後にファイルバッファが設定されます（ＡＦＣ７Ｈ)。結局、ドライブポインタ・ファイルバッファアドレステーブル・ドライブテーブル・ファイルバッファの順でフリーエリアの先頭から確保されます。

スタック領域設定、文字列領域開始、テキスト開始、オープニング表示、フリーバイト数表示はＲＯＭと同じです(ＡＦＦ０Ｈ)。この後もひき続いてディスクコード中で初期化を行います。

ここで４thＲＯＭ初期化処理実行が行われます(８７Ｃ６Ｈ)。ＲＯＭ上では作業域開始アドレスをチェックしていましたが、ここでは取り除かれています。そしてＥＮＤ処理の途中に相当するルーチン（９ＥＣＡＨ）にジャンプします。

最後に行われる初期化は、すべてのファイルをクローズ(９ＥＣＡＨ)、ドライブテーブル初期化（９ＥＤＢＨ)、ＣＬＯＳＥフラグ（ＥＣＡ３Ｈ）クリア、ＮＥＷ処理（４Ｆ０１Ｈ）です（９ＥＣＡ～９Ｆ０７Ｈ)。それから初期テキストをインプットバッファ（Ｅ９Ｂ９Ｈ～）に転送し、ダイレクト実行のための初期値を設定してから実行処理ルーチン（４Ｅ２Ｈ）にジャンプし、初期化処理を終わります。

これらのうち、ＡＦ００Ｈ以降のものはＢＡＳＩＣが起動されるときに破壊されるので、ここを参照するには直接ディスクをアクセスしなければなりません。

第2章

BASICテキストの編集

CONTENTS

2―1　ＢＡＳＩＣプログラム編集の様子

N_{88}－ＢＡＳＩＣにおいて、ＢＡＳＩＣプログラム（以下テキストと略記）の編集は図２－１－１のように行われています。

図の［イ］は、画面上の１行を入力してから、ダイレクトステートメントの実行、あるいは００００Ｈ～７ＦＦＦＨ番地のテキストエリアで削除・格納されるまでの処理を示したものです。

入力した行に行番号がなければダイレクトステートメントとみなして直接実行し、行番号のみならテキストエリアの１行削除。行番号とそれに続く文字があればテキストエリア上の古い行を削除したあとその場所に新しい行の挿入を行います。

また図の［ロ］には、ＤＥＬＥＴＥ文を実行したときのテキストエリアの処理を示してあります。ここではテキストエリアのみを直接変更しますが、その処理はほとんど［イ］のテキストエリアの１行削除の処理の部分と同じです。

2―1―1　ＢＡＳＩＣプログラムの格納状態

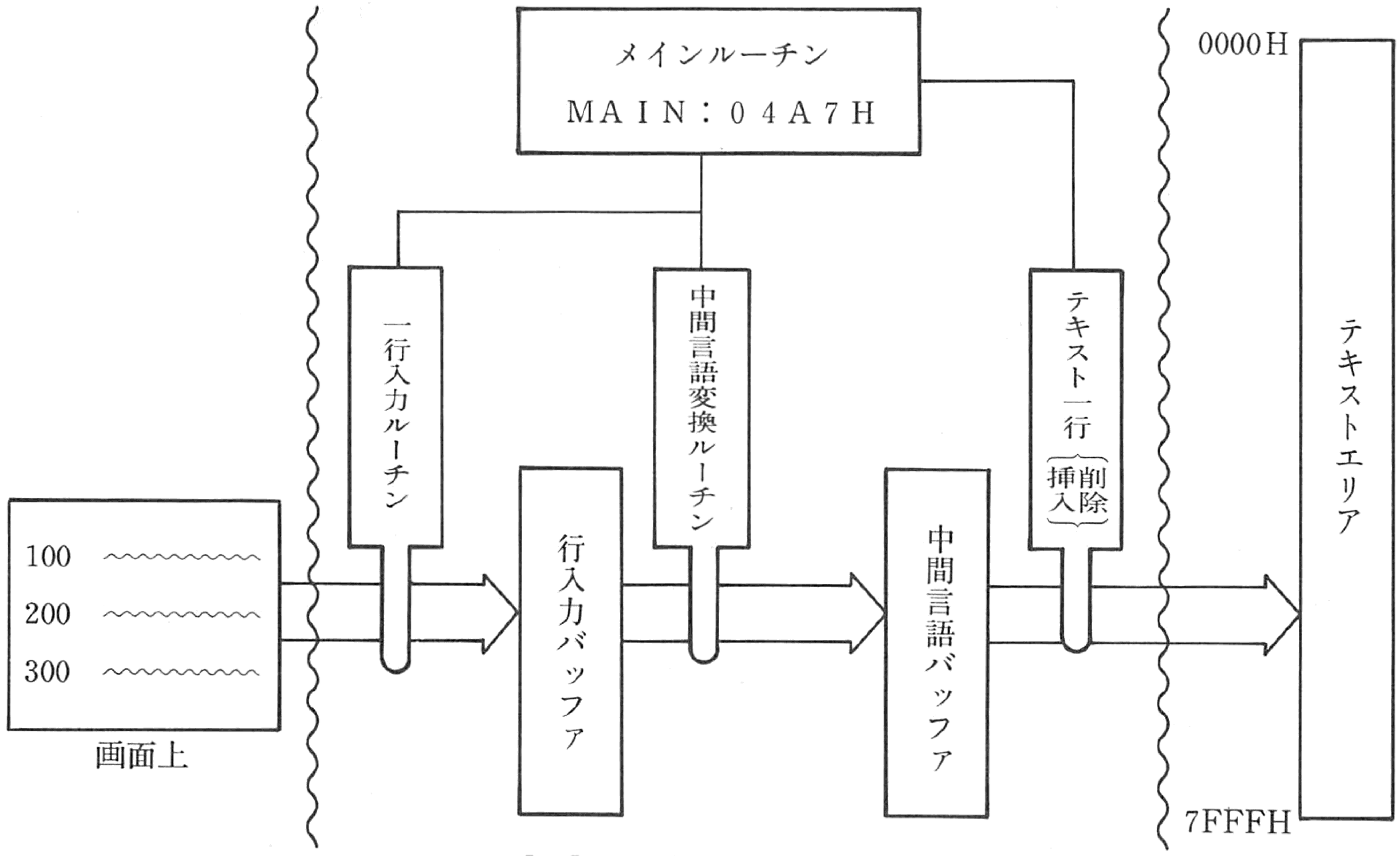
ＢＡＳＩＣのテキストは中間言語に変換後、００００Ｈ～７ＦＦＦＨ番地のテキストエリアに格納されます。中間言語については『２－２中間言語変換』で詳しく述べるとして、ここでは中間言語に変換されたテキストの格納状態について説明しましょう。

テキストは図２－１－２のように格納されています。

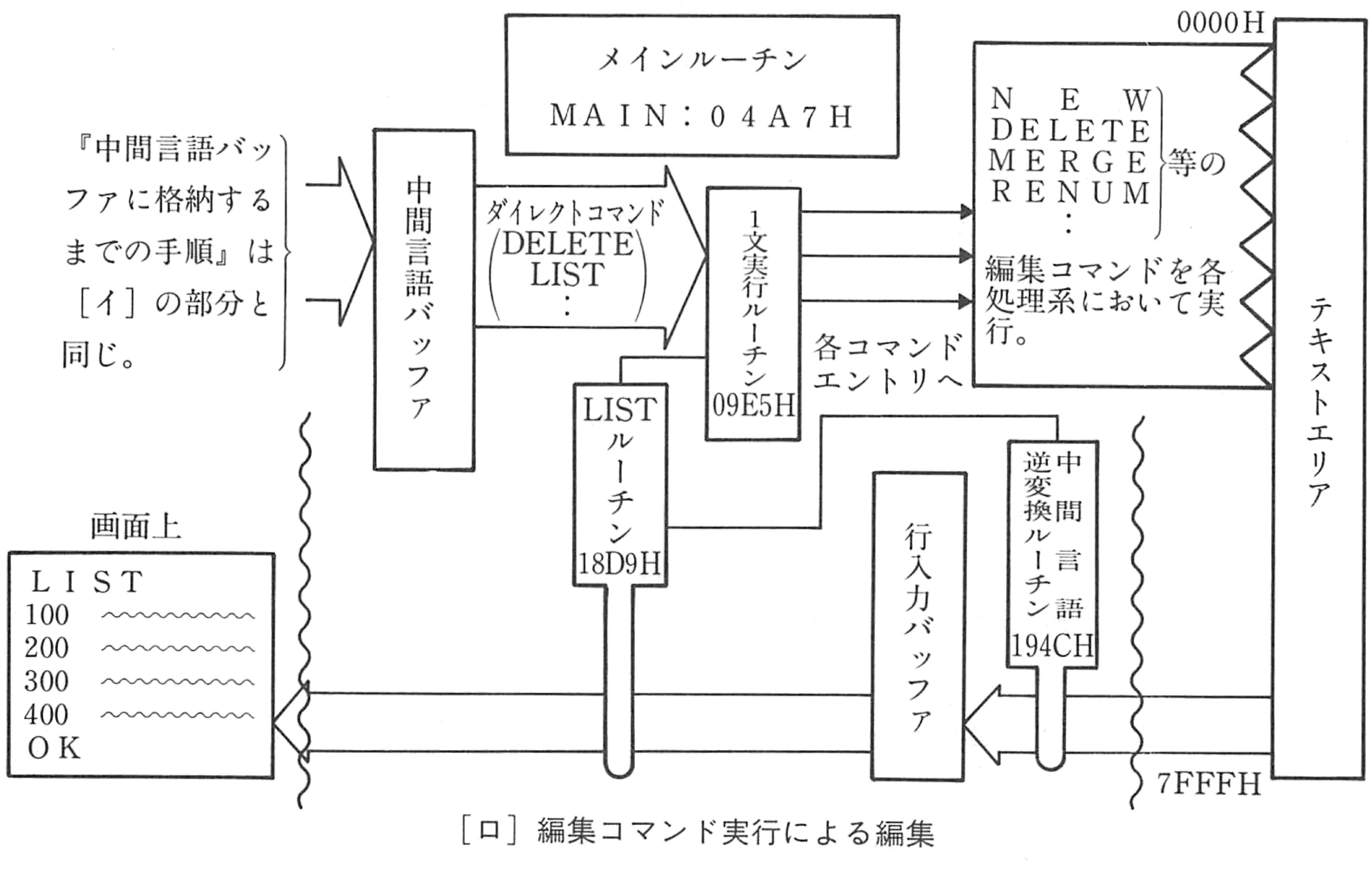
ここで、リンクポインタというのは、行の先頭に置かれていて次の行の先頭アドレスを示すポインタのことです。特に、テキストの最終行のリンクポインタには００００Ｈが入ります。

注）　テキストウィンドウ

N_{88}－ＢＡＳＩＣのテキストは、すべてテキストエリア（００００Ｈ～７ＦＦＦＨ）に格納されます。ところがＢＡＳＩＣのメインＲＯＭ部分も００００Ｈ番地から７ＦＦＦＨ番地までにあるので、このままではＢＡＳＩＣのテキストを読むことができません。これを解決するため、テキストウィンドウと呼ばれる８０００Ｈ番地から８３ＦＦＨ番地までの１Ｋバイトのエリアが用意してあります。テキストエリア中の見たい部分を設定した上で、このウィンドウを通じてＢＡＳＩＣのメインＲＯＭがテキストを自由に読み書きできるようになっているのです。テキストエリア中の見たい領域のアドレスの上位１バイトをポート７０Ｈ番地に出力すると、ハードウェアによってその領域がテキストウィンドウに映されます。また、ポート７８Ｈに出力すれば、その出力した値にかかわらずポート７０Ｈの値をインクリメントできます。



［イ］画面上での直接編集



［ロ］編集コマンド実行による編集

図２－１－１　テキスト処理の概要

```
LIST
100 VARIABLE=100
200 VARIABLE=VARIABLE+1:PRINT VARIABLE
300 GOTO 100
400 END
OK
```

○画面上↑

○テキストエリア↓

アドレス

０００００H～

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
データ	00	11	00	64	00	56	41	52	49	41	42	4C	45	F1	0F	64
読み		0011H		100		V	A	R	I	A	B	L	E	=	整数	100

L.P

0010H

00	34	00	C8	00	56	41	52	49	41	42	4C	45	F1	56	41
Ω	0034H		200		V	A	R	I	A	B	L	E	=	V	A

L.P

0020H

52	49	41	42	4C	45	F3	12	3A	91	20	56	41	52	49	41
R	I	A	B	L	E	+	1	:	PRINT	␣	V	A	R	I	A

0030H

42	4C	45	00	3E	00	2C	01	89	20	0E	64	00	00	44	00
B	L	E	Ω	003EH		300		GOTO	␣	飛び先行番号	100		Ω	0044H	

L.P

0040H

90	01	81	00	00	00
400		END	Ω	0000H	

L.P：テキストの終わり

〔L．Pはリンタポインタの略
Ωは行末の略〕

図２－１－２　テキストの格納例

ＲＯＭ内のルーチンが実際にテキストウィンドウを利用するときには４４Ａ４Ｈ番地、４４Ｂ３Ｈ番地、４４Ｄ５Ｈ番地などにある一連のアドレス変換のルーチンを使っています。

これらのルーチンは、テキストエリア内のアドレスをウィンドウ内のアドレスに変換したり、その逆の変換を行ったりするもので、同時にテキストエリア内のアドレスの上位１バイトを、ポート７０Ｈ番地に読み書きしています。

```
LIST
100 VARIABLE=100
200 VARIABLE=VARIABLE+1:PRINT VARIABLE
300 GOTO 100
400 END
OK
MON

h]070,00
h]D8000,804F
8000 00 11 00 64 00 56 41 52 49 41 42 4C 45 F1 0F 64       d VARIABLE円 d
8010 00 34 00 C8 00 56 41 52 49 41 42 4C 45 F1 56 41      4 ﾈ VARIABLE円VA
8020 52 49 41 42 4C 45 F3 12 3A 91 20 56 41 52 49 41     RIABLE月 :ﾃ VARIA
8030 42 4C 45 00 3E 00 2C 01 89 20 0E 64 00 00 44 00     BLE > , ▮  d  D
8040 90 01 81 00 00 00 00 00 00 00 00 00 00 00 00 00     ┴ ▁
```

図２－１－３　ウィンドウでテキストをのぞく

２―１―２　ＭＡＩＮ：N88―ＢＡＳＩＣの本体

ＢＡＳＩＣプログラムを処理するＲＯＭルーチンの中で、その管理中枢の役割を担っているのが４Ａ７Ｈ番地からのＭＡＩＮルーチンです。このルーチンに中間言語変換や１文実行のルーチンが組み合わさって、ＢＡＳＩＣインタプリタの本体を構成しています。

この節では、ＭＡＩＮルーチンを中心にＢＡＳＩＣインタプリタの処理について説明していきます。

ＭＡＩＮ：４Ａ７Ｈ

領　域	４Ａ７Ｈ～５ＢＣＨ
機　能	N88－ＢＡＳＩＣの処理
ワーク	Ｅ６５６Ｈ：実行中の行番号の値
	Ｅ６ＡＢＨ：現在編集中の行番号の値
	Ｅ６Ｃ９Ｈ：Comunication buffer overflow フラグ
	ＥＡＣＣＨ：文字列エリアの始まり
	ＥＡＦ１Ｈ：フリーエリアの終わり
	ＥＢ００Ｈ：ＡＵＴＯモードフラグ
	ＥＢ０１Ｈ：ＡＵＴＯモードで次に発生すべき行番号
	ＥＢ０３Ｈ：ＡＵＴＯモード時の増分
	ＥＢ０５Ｈ：１文実行前のテキストポインタ
	ＥＢ１０Ｈ：汎用ワークエリア

ＥＢ１８Ｈ：テキストエンド＋１
ＥＣ８８Ｈ：カレントファイルバッファのアドレス

破　壊　Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ

入　力　なし

出　力　ポート７１Ｈ：ＲＯＭバンクの設定

ルーチン　ＲＳＴ１０Ｈ：１文字読み込み
ＲＳＴ１８Ｈ：画面へ１文字出力
ＲＳＴ２０Ｈ：レジスタＨＬとＤＥの比較
０３９３Ｈ：Syntax error 処理ルーチン
０５Ｃ１Ｈ：リンクポインタを書き直す
０６０５Ｈ：レジスタＤＥと同じ行番号の文を捜す
０６３２Ｈ：テキストを中間言語に変換する
０８ＡＦＨ：テキスト中の読み飛ばしたブランクの分だけポインタを戻す
０Ｂ３４Ｈ：行番号か行アドレスを読み込んでレジスタＤＥに出力する
０Ｃ３ＣＨ：Undefined line number エラー処理ルーチンへ
１Ｃ８１Ｈ：行アドレス　→　行番号変換
２０Ｆ２Ｈ：ＩＮＣ　ＨＬ
２８Ｃ２Ｈ：ＨＬの数値を画面に出力する
４４Ａ４Ｈ：実アドレスＨＬ　→　ウィンドウ内アドレス
４４Ｂ３Ｈ：ウィンドウ内アドレスＢＣ　→　実アドレスＢＣ
４４Ｄ５Ｈ：ウィンドウ内アドレスＨＬ　→　実アドレスＨＬ
４４Ｆ１Ｈ：テキスト移動
４５２ＣＨ：テキスト移動
４Ｃ７０Ｈ：ＥＣ８８Ｈ番地が０かどうかをチェックし、０の場合には
１文実行ルーチン（９９６Ｈ番地）へ飛ぶ
４Ｆ２１Ｈ：変数のクリア
５Ｆ９３Ｈ：画面上の１行を入力する
７ＣＣＤＨ：Line buffer overflow エラー処理
ＥＤ２４Ｈ：フック、ＪＰ　８ＤＥ５Ｈ：ＣＨＡＩＮ文フラグのチェック
ＥＤ２ＡＨ：フック、ＲＥＴ
ＥＤ３０Ｈ：フック、ＲＥＴ
ＥＤ３ＦＨ：フック、ＲＥＴ
ＥＤＡ８Ｈ：フック、ＪＰ　ＡＤ５１Ｈ：プロテクトチェックルーチン

動　作　ＭＡＩＮルーチンは次の４つの処理から成り立っています。

【処理１】ＡＵＴＯモード（ＡＵＴＯコマンドを実行したときのモード）のときの画面への行番号出力
【処理２】画面からの１行入力
【処理３】行番号読み込みおよび中間言語変換
【処理４】ダイレクトコマンドの実行もしくはテキストエリアの編集

それでは、それぞれの処理について説明を加えていきましょう。

【処理１】　ＡＵＴＯモードのときの画面への行番号出力

まず、ポート７１ＨにＦＦＨを出力して、６０００Ｈ～７ＦＦＦＨの4thROMの領域にメインROMを指定します。

次に、実行中の行番号を格納するＥ６５６Ｈ番地にＦＦＦＦＨを入れた後、ラインバッファのオーバーフローを示すフラグ（Ｅ６Ｃ９Ｈ）を見て、フラグが立っていた場合には、エラー処理のルーチンへ飛びます。

そして、ＡＵＴＯモードを示すフラグ（ＥＢ００ＨＨ番地）を見ます。このフラグは、ＡＵＴＯコマンドが実行されると、『^Ｃ』（ＳＴＯＰキー）が押されるまで立っているフラグです。フラグが立っていない場合には【処理２】に飛び、立っている場合には以下の処理を行います。

① まず、ＡＵＴＯモードで表示する行番号を納めているワーク（ＥＢ０１Ｈ～ＥＢ０２Ｈ番地）からデータを読み、２８Ｃ２Ｈ番地のルーチンを用いて画面に出力します。２８Ｃ２Ｈ番地のルーチンでは2バイトの数値を出力形式（数字の文字列）に変換してから画面に出力しています。

② 次に、６０５Ｈ番地のルーチンを用いてテキスト内の行番号をサーチし、該当する行番号が存在するかどうかを確かめます。存在する場合には画面上に出力した行番号の後に『＊』を付け加え、存在しない場合には『＊』の代わりにスペース『␣』を付け加えます。

③ 最後に、上記の『＊』なり『␣』なりをＥＢ００Ｈ番地に出力した後、【処理２】に移ります。

【処理２】画面からの１行入力

これは５Ｆ９２Ｈ番地のルーチンを用いるものです。このルーチンは、リターンキー（^M）かＳＴＯＰキー（^Ｃ）が押されたときに、画面上のカーソルがある場所の１行分の文字列（最大254バイト）を、Ｅ９Ｂ９Ｈ～ＥＡＢＡＨ番地の行入力バッファに転送する機能を持ちます。ここで、ＳＴＯＰキー（^Ｃ）が押されていた場合には、ＡＵＴＯフラグをクリアして、ＭＡＩＮの最初の処理に戻ります。

５Ｆ９２Ｈ番地のルーチンの詳細は、既刊「ＰＣ８８０１　N_{88}－ＢＡＳＩＣ解析マニュアル（川村清　著）」に掲載されていますので、そちらを参照して下さい。

【処理3】行番号読み込みおよび中間言語変換

ここではまず、１０Ｈ番地のルーチンを用いて行入力バッファから１文字だけ読み込んで文字の種類をチェックします。チェックする種類は数字（３０Ｈから３９Ｈまで）かどうかであり、チェックの結果は保存され、【処理4】で行番号付きのプログラムとダイレクトコマンドとの区別に用いられます。特に、その文字が００Ｈだった場合、すなわち行入力バッファが空だった場合には、ＭＡＩＮの最初の処理に戻ります。

次に、Ｂ３ＦＨ番地のルーチンを用いて、行入力バッファから行番号だけの読み込みを行います。読み込まれた行番号は、バイナリ形式でレジスタＤＥに入り、【処理4】のテキスト編集に使われます。行番号がないときは、Ｂ３ＦＨ番地のルーチンは何もしません。

最後に６３２Ｈ番地のルーチンを用い、行入力バッファ中の行番号を除いた残りの部分を中間言語に変換して、Ｅ８７ＡＨ～Ｅ９Ｂ７Ｈ番地の中間言語バッファに格納します。６３２Ｈのルーチンについては、２－２－２節を参照して下さい。

【処理4】ダイレクトコマンドの実行もしくはテキストエリアの編集

ここでは、【処理2】で入力した1行がダイレクトコマンドか行番号付きのプログラムかどうかを調べた後、ダイレクトコマンドであれば1文実行のルーチンへ飛び、プログラムであればテキストエリアの編集を行います。

まず、【処理3】の最初のＲＳＴ１０Ｈを用いた1文字読み込みで、先頭の文字が数字か否かは確認してあるので、そこからダイレクトステートメントかプログラムかを判断し、ダイレクトステートメントであれば９Ｅ５Ｈ番地の1文実行のルーチンに飛び、処理を移します。９Ｅ５Ｈ番地のルーチンの詳細については、３－１－３を参照して下さい。

行番号付きのプログラムのときは、さらに以下の順番で処理を行います。

① まずＡＤ５１Ｈ番地のルーチンでプロテクトチェックを行います。これはＥＣ２９Ｈ番地のプロテクトフラグを調べて立っている場合にはＢ６Ｈ番地に飛んでIllegal function callのエラー処理を行う、というものです。

② その後、【処理3】で読み込んだ行番号を調べ、値が0であれば３９３Ｈ番地のSyntax errorの処理に飛びます。

③ 次にＡＵＴＯモードの増分をＥＢ０３Ｈ番地から持ってきて、【処理3】で読み込んだ行番号を足し、それをＡＵＴＯモードの次の行番号としてＥＢ０１番地に入れます。これはＡＵＴＯのときでなくても必ず行う処理です。

④ 次に６０５Ｈ番地のルーチンを用いて、【処理3】で入力したものと同じ行番号を持つ行がテキスト内に存在しているかどうかを調べます。ここで、処理の条件が4種類に分けられます。

第１に、同じ行番号を持つ行が存在せず、行入力バッファ内に行番号以外の文字がない場合。このときには、Ｃ３ＣＨ番地に飛び、Undefined line numberのエラー処理を行います。

第２に、同じ行番号を持つ行が存在せず、行入力バッファ内に行番号以外の文字が存在する場合。この場合には、中間言語バッファ中の新しい行のサイズだけテキスト領域を開けるために４４Ｆ１Ｈ番地のルーチンを用いてテキストの移動を行います。そして空いたテキスト領域に中間言語バッファから新しいテキストを転送し、次のリンクポインタの処理へ進みます。

第３に、同じ行番号を持つ行が存在し、行入力バッファ内に行番号以外の文字が存在しない場合。この場合には、４５２ＣＨ番地のルーチンを用いて、同じ行番号を持つ古い行をテキスト領域から削除し、後続のテキストを移動して削除した部分を埋めた後、リンクポインタの処理に移ります。

第４に、同じ行番号を持つ行が存在し、行入力バッファ内にも行番号以外の文字が存在する場合。ここではまず、第３の場合と同じく４５２ＣＨ番地のルーチンを用いて古い行を削除しておき、そのあとに第２の場合と同じく４４Ｆ１Ｈ番地のルーチンを用いてテキスト領域を空けて、中間言語バッファから新しいテキストを転送した後、リンクポインタ処理に移ります。

以上４通りの処理を終わると、第１の場合を除き、処理の最後に５Ｃ１Ｈ番地のルーチンを用いて新しいリンクポインタをテキスト中のすべての行に書き込み、さらに４Ｆ２１Ｈ番地のルーチンで変数のクリアを行ったあと、【処理１】の先頭に戻ります。

2－2　中間言語変換

画面上で編集するか、ディスクあるいはテープなどからロードするかして得られたＢＡＳＩＣプログラム（以下「テキスト」と略記）は中間言語の形式で００００Ｈ～７ＦＦＦＨ番地のテキストエリアに格納されています。

ここでは、中間言語の種類と構造、ならびに、中間言語変換処理の実際についての説明を行います。

2－2－1　中間言語と中間言語テーブル

中間言語には図２－２－１のような種類があります。

このうち、８０ＨからＦＦＨまでは予約語に対応しており、８０ＨからＦＥＨまでは１文字で予約語一語を、またＦＦＨは２バイトコード用となっていて次の１文字と組み合わせて合計２文字で予約語一語を表しています。

リストから中間言語へ変換するとき、予約語に関しては、必ずしも予約語テーブルにあるとりの１文字あるいは２文字で規定されるとは限りません。例えば、『ＷＨＩＬＥ』（ＡＦＨ）のあとには必ず『＋』（Ｆ３Ｈ）を付加して中間言語のテキストに変換されます。また、リスト上の『ＲＥＭ』に対応する中間言語は８ＦＨのみであるのに対して、『’』（シングルクォートによる注釈）では３９Ｈ、８ＦＨ、Ｅ９Ｈ（＝『：ＲＥＭ♥』）となります。ここで、Ｅ９Ｈというのは、中間言語で表した『’』のことです。このため、『：ＲＥＭ♥』を入力してリストをとると『’』に変わってしまいます。つまり、『’』は『ＲＥＭ』と比較して、なんと２バイトも多いのです!!

このように見てみると、コメント文では、『’』を使えば見た目がきれいになるのに対して、『ＲＥＭ』を使えば実行がわずかながら速くなる、ということもわかると思います（もっとも、速度に関しては、実際に目に見えるほどの違いは決して出てこないといってもよい位ですが……）。

2－2－2　中間言語変換処理（６３２Ｈ）

中間言語変換処理は、主に６３２Ｈ番地のルーチンで行われています。このルーチンは、５０３Ｈ番地（ＭＡＩＮルーチン内）と７５Ｃ９Ｈ番地（ＴＥＲＭ処理ルーチン内）からＨＬをポインタにしてコールされ、ＬＩＳＴ形式のテキストを中間言語形式に変換するものです。

ここでは、この６３２Ｈ番地のルーチンを重点的に説明します。

<table>
<tr><th>コード</th><th>意　味</th><th>説　　　明</th><th>備　　　考</th></tr>
<tr><td>0 0</td><td>行　末</td><td>中間言語一行の区切り。</td><td></td></tr>
<tr><td>0 1
〜
0 6</td><td>———</td><td>未使用。</td><td></td></tr>
<tr><td>0 7</td><td>BELL</td><td rowspan="4">画面上にリストした際、本来の制御コードとして働く。</td><td rowspan="4">通常、ＬＦ以外はテキスト上に存在しない。</td></tr>
<tr><td>0 8</td><td>ＢＳ</td></tr>
<tr><td>0 9</td><td>ＴＡＢ</td></tr>
<tr><td>0 A</td><td>ＬＦ</td></tr>
<tr><td>0 B</td><td>＆O</td><td>８進定数２バイトが続く。</td><td rowspan="3">テキスト上の次に続く文字の意味を規定している（０ＦＨや、１ＣＨ、１ＤＨ、１ＦＨも同様）</td></tr>
<tr><td>0 C</td><td>＆H</td><td>16進定数２バイトが続く。</td></tr>
<tr><td>0 D</td><td>アドレス</td><td>飛び先アドレス２バイトが続く。</td></tr>
<tr><td>0 E</td><td>行番号</td><td>飛び先行番号２バイトが続く。</td><td rowspan="4">整数定数は１桁あるいは２桁で用いられることが多いので、このように３つの表現法が用意されており、高速化に役立っている（なお、コード１ＢＨは『10』を表す定数としてキープされているが、実際には生成されない）。</td></tr>
<tr><td>0 F</td><td>整数</td><td>「10」〜「255」の整数1バイトが続く。</td></tr>
<tr><td>1 1
〜
1 B</td><td>整数</td><td>１ケタの整数を表す。
（例．１１Ｈは『０』
１ＡＨは『９』）</td></tr>
<tr><td>1 C</td><td>整数</td><td>整数定数２バイトが続く。</td></tr>
<tr><td>1 D</td><td>単精度</td><td>単精度実数２バイトが続く。</td><td></td></tr>
<tr><td>1 E</td><td>———</td><td>テキスト上では未使用。</td><td>ルーチン内部では使われるがテキストに存在することはない。</td></tr>
<tr><td>1 F</td><td>倍精度</td><td>倍精度実数４バイトが続く。</td><td></td></tr>
<tr><td>2 0
〜
7 F</td><td>文字</td><td>ラベルや変数名に用いられ、キャラクタコードを直接使っている。</td><td>アルファベットの小文字は大文字に変換されるため、文定列中を除いては使われない。</td></tr>
<tr><td>8 0
〜
F F</td><td>予約語</td><td>コマンドや関数等の予約語を表す。</td><td>詳細は、次の図２－２－２を参照。</td></tr>
</table>

図２－２－１　中間言語の種類

	80H～8FH	90H～9FH	A0H～AFH	B0H～BFH	C0H～CFH	D0H～DFH	E0H～EFH	F0H～FFH
0	――――	STOP	TRON	WEND	CLOSE	PRESET	USR	>
1	END	PRINT	TROFF	CALL	LOAD	PAINT	FN	=
2	FOR	CLEAR	SWAP	――――	MERGE	TERM	SPC (	<
3	NEXT	LIST	ERASE	――――	FILES	SCREEN	NOT	+
4	DATA	NEW	EDIT	――――	NAME	BLOAD	ERL	−
5	INPUT	ON	ERROR	WRITE	KILL	BSAVE	ERR	*
6	DIM	WAIT	RESUME	COMMON	LSET	LOCATE	STRING$	／
7	READ	DEF	DELETE	CHAIN	RSET	BEEP	USING	^
8	LET	POKE	AUTO	OPTION	SAVE	ROLL	INSTR	AND
9	GO␣TO	CONT	RENUM	RANDOMIZE	LFILES	HELP	'	OR
A	RUN	OUT	DEFSTR	DSKO$	MON	――――	VARPTR	XOR
B	IF	LPRINT	DEFINT	OPEN	COLOR	KANJI	ATTR$	EQV
C	RESTORE	LLIST	DEFSNG	FIELD	CIRCLE	TO	DSKI$	IMP
D	GOSUB	CONSOLE	DEFDBL	GET	COPY	THEN	SRQ	MOD
E	RETURN	WIDTH	LINE	PUT	CLS	TAB (	OFF	¥
F	REM	ELSE	WHILE	SET	PSET	STEP	INKEY$	2バイトコード

図2-2-2①　中間コード対応表・2の1（予約語1バイトコード）

	80～8F	90～9F	A0～AF	B0～BF	C0～CF	D0～DF	E0～EF	F0～FF
0	―――	INP	CVI	―――	―――	DSKF	ISET	―――
1	LEFT$	POS	CVS	―――	―――	VIEW	IEEE	―――
2	RIGHT$	LEN	CVD	―――	―――	WINDOW	IRESET	―――
3	MID$	STR$	EOF	―――	―――	POINT	STATUS	―――
4	SGN	VAL	LOC	―――	―――	CSRLIN	CMD	―――
5	INT	ASC	LOF	―――	―――	MAP	―――	―――
6	ABS	CHR$	FPOS	―――	―――	SEARCH	―――	―――
7	SQR	PEEK	MKI$	―――	―――	MOTOR	―――	―――
8	RND	SPACE$	MKS$	―――	―――	PEN	―――	―――
9	SIN	OCT$	MKD$	―――	―――	DATE$	―――	―――
A	LOG	HEX$	―――	―――	―――	COM	―――	―――
B	EXP	LPOS	―――	―――	―――	KEY	―――	―――
C	COS	CINT	―――	―――	―――	TIME$	―――	―――
D	TAN	CSNG	―――	―――	―――	WBYTE	―――	―――
E	ATN	CDBL	―――	―――	―――	RBYTE	―――	―――
F	FRE	FIX	―――	―――	―――	POLL	―――	―――

図２−２−２②　中間コード対応表・２の２（予約語２バイトコード：ＦＦＸＸ）

中間言語変換：６３２Ｈ

領　域　６３２Ｈ～８ＢＥＨ（mkII、ＳＲでの変更箇所なし）

機　能　テキストを中間言語形式に変換する

ワーク　ＥＡＢＥＨ～ＥＡＢＦＨ：中間言語変換用ワークエリア

破　壊　Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ

入　力　ＨＬ：テキストポインタ
６Ｂ５６Ｈ～６Ｂ８９Ｈ：予約語のインデックス
６Ｂ８ＡＨ～６Ｅ８０Ｈ：予約語テーブル
Ｅ９Ｂ９Ｈ～ＥＡＢＡＨ：行入力バッファ
ＥＡＢＤＨ：変数の種類
ＥＣ４１Ｈ～ＥＣ４４Ｈ：ＦＡＣ（単精度型４バイト）

出　力　ＢＣ：中間言語形式でのテキスト１行の文字数
ＨＬ：テキストポインタ
Ｅ８７ＡＨ～Ｅ９Ｂ７Ｈ：中間言語バッファ

ルーチン　ＲＳＴ１０Ｈ：テキストからの１文字読み込み
０３Ｂ３Ｈ：エラー処理ルーチン
０８５ＦＨ：（ＤＥ）に数値３ＡＨを書き込んで、カウンタを更新し、０になったら Line buffer overflow エラー
０８６１Ｈ：Ａｃｃの値を（ＤＥ）に書き込んだあと、カウンタを更新し、０になったら　Line buffer overflow　エラー
０Ｂ３４Ｈ：行位置の読み込み（→ＤＥ）
１４１４Ｈ：（ＨＬ）から１文字読み込み、英小文字だったら大文字に変換する
１４２３Ｈ：テキストから16進数・８進数（&Ｈ・&Ｏ）を読み込む
１ＡＢ４Ｈ：（ＨＬ）の文字の種類のチェック
２６ＢＣＨ：数値を表す文字列を単精度数としてＦＡＣに入力
５２１６Ｈ：（ＨＬ）が大文字か否かのチェック
ＥＤ１ＥＨ：フック、ＲＥＴ
ＥＤ２１Ｈ：フック、ＲＥＴ
ＥＤ５１Ｈ：フック、ＲＥＴ
ＥＤ５４Ｈ：フック、ＲＥＴ
ＥＤ５７Ｈ：フック、ＲＥＴ

動　作　行入力バッファ（Ｅ９Ｂ９Ｈ～ＥＡＢＡＨ）内の、ＨＬが指している箇所から行末を表すマーク００Ｈがあるところまでの部分を中間言語に変換し、中間言語バッファ（Ｅ８７ＡＨ～Ｅ９Ｂ７Ｈ）に出力します。

このルーチンは、次のように処理を行っています。

【処理１】　初期化および行末のチェック・処理　（６３２Ｈ～６５８Ｈ）

ワークエリアのＥＡＢＥＨとＥＡＢＦＨに０を代入し、レジスタＢＣとＤＥに初期値（それぞれ１３ＢＨとＥ８７ＡＨ）を与えます。ここでＢＣは文字数のカウンタ、ＤＥは、中間言語バッファへ書き込むポインタとして用いられます。

初期化の部分はここまでで、この先からメインループを構成しています。

ＨＬをポインタとして、行入力バッファから１バイト（１文字）をとってきて、それが０かどうかをチェックします。０の場合には、行末とみなしＤＥをポインタとして中間言語バッファに０を３バイト分書き込み、ＨＬに「中間言語バッファの先頭アドレス－１」を代入して終了します。

０でない場合には、次の【処理２】へ進みます。

【処理２】　文字識別その１　（６５９Ｈ～６９５Ｈ）

ここでは、前の処理で入力した文字が『”』（ダブルクォート：２２Ｈ）あるいは『␣』（スペース：２０Ｈ）かどうかを調べ、そのどちらでもない場合、さらに、ＥＡＢＥＨが０かどうかを調べます。ＥＡＢＥＨは、文字列内かどうか、すなわち中間言語に変換する必要がないかどうかを表すフラグ（文字列内すなわち変換の必要がない場合には０でない値が入る）です。そこでＥＡＢＥＨが０だった場合は次の【処理３】に進みます。前の処理で入力した文字は、ここでは保存されています。

（１）　前の処理で入力した文字が『”』だった場合、もう一度『”』が来るか、行末の０が来るまでの間、（ＨＬ）から文字を読み込み、そのまま（ＤＥ）へ書き込みます。この間、ＤＥとＨＬをインクリメントし、ＢＣをデクリメントすることは言うまでもありません。ここで『”』が来たときには次の【処理３】に移り、０が来たときには行末とみなして【処理１】の行末処理に移ります。

（２）　前の処理で入力した文字が『␣』だった場合、そのまま『␣』を中間言語バッファに１文字出力した後、【処理１】の行末チェックに戻ります。

（３）　最後にＥＡＢＥＨが０でなかった場合（すなわち中間言語に変換しない場合）、前の処理で入力した文字をまず中間言語バッファに１文字出力した後、その文字が『：』（コロン：３ＡＨ）かどうかをチェックして、コロンだった場合にはＥＡＢＥＨとＥＡＢＦＨに０を代入した後、【処理１】の行末チェックに戻り、コロンでなかった場合には、さらにその文字が８４Ｈあるいは８ＦＨかどうかを見て、どちらでもなかった場合には何もせずに【処理１】の行末チェックに戻ります。ここで、８４Ｈは中間言語の『ＤＡＴＡ』、８ＦＨは『ＲＥＭ』に相当します。８４Ｈだった場合には、ＥＡＢＥＨとＥＡＢＦＨに１を代入した後、【処理１】の行末チェックに戻ります。８ＦＨだった場合には、行入力バッファ中の後続の文字列を、行末の０が来

るまでの間、そのままの形で中間言語バッファに転送します。この際、行末の０が現れた場合には【処理１】の行末処理へ飛びます。

【処理３】　文字識別その２　（６９６Ｈ～６ＥＡＨ）

ここでは、前の処理で入力した文字の中から、８０Ｈ以上・３ＦＨ・英文字・それ以外（数字や記号等）の各種により分け、それぞれの処理ルーチンに飛ばす処理を行っています。

（１）　まず、文字が８０Ｈ以上かどうかをチェックし、８０Ｈ以上ならそのまま【処理１】の行末チェックに戻ります。８０Ｈ以上の文字を読み飛ばすわけです。

（２）　次は、『？』（３ＦＨ）かどうかのチェックで、『？』の場合には【処理５】に飛びます。このとき、【処理５】から先では、ＥＡＢＦＨに０を代入して中間言語バッファに９１Ｈ（中間言語のＰＲＩＮＴ）を出力した後、【処理１】の行末チェックに戻ります。

（３）　続いて、英小文字を大文字に変換した後、英文字かどうかをチェックして、英文字でない場合には【処理６】に飛びます。英文字の場合には、以後の文字列が特に『ＧＯ　ＳＵＢ』であるかどうかのチェックだけを行い、『ＧＯ　ＳＵＢ』の場合には【処理５】に飛び、ＥＡＢＦＨに「１」を、中間言語バッファに８ＤＨ（中間言語のＧＯＳＵＢ）をそれぞれ出力してから【処理１】の行末チェックに戻ります。この処理は『ＧＯ　ＳＵＢ』を『ＧＯＳＵＢ』として扱うためのものです。『ＧＯ　ＳＵＢ』でない場合には【処理４】の予約語チェックに移ります。

なお、『ＧＯ　ＴＯ』と『ＧＯＴＯ』については、両方とも予約語テーブルに登録されていて、『ＧＯＳＵＢ』のような面倒な処理を行わずに済んでいます。もっとも、ＰＣ８８０１が発表された当初のVer 1.0のＲＯＭでは、『ＧＯＳＵＢ』と同様に『ＧＯＴＯ』の場合も２種類の書き方を許すための処理があって、それがバグの原因にもなっていたようです。

【処理４】予約語のチェック・処理　（６ＥＢＨ～７５ＣＨ）

ここではまず、行入力バッファから先頭の１文字をとってきて、文字列の先頭とみなし、インデックスを用いて予約語テーブルのアドレスを割り出します。予約語テーブルとはアルファベット順に並んでいる予約語とその中間言語コードとの対応表のことであり、また予約語インデックスとは予約語テーブルの目次に相当するもので、これらはそれぞれメインＲＯＭ上の６Ｂ８Ａ～６Ｅ８０Ｈと６Ｂ５６Ｈ～６Ｂ８９Ｈにあって図２－２－３のような形式になっています。例えば『ＡＵＴＯ』ならＡの領域上（６Ｂ８ＡＨ～６Ｂ９ＦＨ）の最初の予約語でダンプをみると『ＵＴマ』＋Ａ８Ｈ　というぐあいです。ここで最初の１文字（ＡＵＴＯでは『Ａ』）が何であるかはその場所によって分かっているので、テーブル内では省略されています。

この予約語テーブルと行入力バッファ上の文字列とを比較して一致しない場合には【処理８】に進みます。一致する場合、文字列の最後の文字が『(』であるかどうか、または、対応する中間言語コードがＥ１Ｈ（ＦＮ）あるいはＥ０Ｈ（ＵＳＲ）かどうかをチェックします。そのい

［イ］　予約語インデックス

A：	6B8AH	N：	6D40H
B：	6BA0H	O：	6D4FH
C：	6BAFH	P：	6D68H
D：	6C0EH	Q：	6D97H
E：	6C4AH	R：	6D98H
F：	6C6FH	S：	6DDAH
G：	6C89H	T：	6E21H
H：	6C9BH	U：	6E41H
I：	6CA4H	V：	6E4AH
J：	6CCEH	W：	6E58H
K：	6CCFH	X：	6E7BH
L：	6CDCH	Y：	6E7FH
M：	6D1CH	Z：	6E80H

［ロ］　予約語テーブル

6B8AH：	55　54　CF　A8	：(A) U　T [O]
	4E　C4　F8	：(A) N [D]
	42　D3　06	：(A) B [S]
	54　CE　0E	：(A) T [N]
	53　C3　15	：(A) S [C]
	54　54　52　A4　EB	：(A) T　T　R [$]
	00	：予約語『A…』終了

［ハ］　構成

```
6B8AH   55  54  CF  A8
        ↓   ↓   ↓   ↓    中間言語　1バイトコード
    41  55  54  4F：A8
    A   U   T   O
```

予約語の最後の文字はその最上位ビットを立ててあるので、80H以上の文字が現れたならば、その直後の1バイトは中間コードを表現していることが容易にわかります。

```
6B91H   42  D3  06
        ↓   ↓   ↓ ↘      中間言語　2バイトコード
    41  42  53：FF  86   （2バイト目の最上位ビットを反転し
    A   B   S              て1バイトに短縮する）
```

図2−2−3　予約語テーブル

ずれでもないときには、さらに行入力バッファから１文字とってきてその文字が英数字かピリオド『.』である場合に、やはり【処理８】に進みます。

それ以外については、中間言語コードが１バイトコード（８０H以上）であるか２バイトコード（８０H未満）であるかをチェックして、１バイトコードであれば次の【処理５】へ移ります。また、２バイトコードであれば８０Hを足した後、ＦＦＨを前に付けて中間言語バッファに出力し、ＥＡＢＦH番地に０を代入して、【処理１】の行末チェックに戻ります。

【処理５】行番号の続く予約語のチェック　（７５ＤＨ～７ＡＣＨ）

ここでは、予約語に対応する１バイトコードの中間言語の中で、その後に行番号が続けられるものかどうかをチェックします。行番号の続く予約語は、図２－２－４のとおり15種類で、７７０H～７７ＦHのテーブルに中間言語のみの形で納められています。

ＡＵＴＯ	Ａ８Ｈ	ＤＥＬＥＴＥ	Ａ７Ｈ	ＥＤＩＴ	Ａ４Ｈ
ＥＬＳＥ	９ＦＨ	ＥＲＬ	Ｅ４Ｈ	ＧＯＴＯ	８９Ｈ
ＧＯＳＵＢ	８ＤＨ	ＬＩＳＴ	９３Ｈ	ＬＬＩＳＴ	９ＣＨ
ＲＥＮＵＭ	Ａ９Ｈ	ＲＥＳＴＯＲＥ	８ＣＨ	ＲＥＳＵＭＥ	Ａ６Ｈ
ＲＥＴＵＲＮ	８ＥＨ	ＲＵＮ	８ＡＨ	ＴＨＥＮ	ＤＤＨ

図２－２－４　行番号の続く予約語とその中間言語

このテーブルをもとにして行番号の続く予約語かどうかをチェックし、そうであれば「１」を、そうでなければ「０」をＥＡＢＦHに出力します。

この後、特殊コードのチェックが続きます。まず、９ＦＨ（ＥＬＳＥ）かどうかを調べ、もし９ＦＨなら、あらかじめコロン『：』を中間言語バッファに出力しておきます。次に、ＡＦＨ（ＷＨＩＬＥ）かどうかを見て、ＡＦＨのときにはこのコードの後に『＋』を続けて中間言語バッファに出力します。そして最後に、Ｅ９Ｈ（『'』）かどうかチェックして、Ｅ９Ｈの場合には、まず『：』と８ＦＨ（ＲＥＭ）をＥ９Ｈの前に挿入し、以後の文字列をコメントとしてＥ９Ｈの後にそのままの形で、中間言語バッファに出力します。

以上の３種類の特殊コードのチェックを終えた後、Ｅ９Ｈの場合以外は【処理７】の＆定数（＆Ｈ・＆Ｏ）の処理の部分に進みます。

【処理６】　数字の処理　（７ＡＤＨ～８２ＤＨ）

ここへは【処理３】から飛んできていて、まずＡレジスタ上の１文字が数字かどうかのチェックをし、数字でないときにはさらに【処理７】に進みます。

数字であるときには、以下の順序で処理を行います。

（1）　ＥＡＢＦＨ番地の値が８０Ｈより大きかった場合、【処理２】内の中間言語に変換をしない場合の処理に移ります。

（2）　ＥＡＢＦＨ番地が値が００Ｈの場合、テキストから読んだ文字は行の先頭の行番号以外の数値であるとみなして次の処理を行います。

①　２６ＢＣＨ番地のルーチンをコールして、行入力バッファ上の文字列を数値データに変換してＦＡＣに格納します。このときＥＡＢＤＨ番地には数の型を示す数値（０８Ｈ、０４Ｈ、または０２Ｈ）が入ります。また、２６ＢＣＨ番地のルーチンでブランク（スペース・タブ・ラインフィード等）を読み飛ばしてしまっているので、その後で８６１Ｈ番地のルーチンを使って、読み飛ばした分だけもとに戻します。

②　ＥＡＢＤＨ番地の値とＦＡＣの値から、次の図２－２－５の処理を行った後、【処理１】の行末チェックに戻ります。

ＥＡＢＤＨの値	数値の大きさ	処　　理
2（整数）	０～９	数値に１１Ｈを足して、１１Ｈ～１ＡＨまでのコードに変換する。
	１０～２５５	コード０ＦＨを付け加え、ＦＡＣの下位１バイトのみを出力する。（合計2バイト）
	－３２７６８～－１ および ２５６～３２７６７	コード１ＣＨを付け加え、ＦＡＣのＥＣ４１Ｈ～ＥＣ４２Ｈの2バイトをそのままテキストに出力する。
4（単精度）	仮数部3バイト ＋ 指数部1バイト	コード１ＤＨを付け加え、ＦＡＣのＥＣ４１Ｈ～ＥＣ４４Ｈの4バイトをそのままテキストに出力する。
8（倍精度）	仮数部7バイト ＋ 指数部1バイト	コード１ＦＨを付け加え、ＦＡＣのＥＣ３ＤＨ～ＥＣ４４Ｈの8バイトをそのままテキストに出力する。

図２－２－５　定数の変換

（３）　ＥＡＢＦＨ番地の値が０１Ｈ～７ＦＨで、行入力バッファ上の文字が２ＥＨ（ピリオド『.』）の場合には、ＥＡＢＦＨ番地の値が８０Ｈよりも大きい場合と同様に、【処理２】の、中間言語に変換をしない場合の処理に飛びます。
（４）　ＥＡＢＦＨ番地の値が８０Ｈ以上でも００Ｈでもなく、文字が２ＥＨでない場合には、文字が行番号であるとみなし、まず０ＥＨ（中間言語で、以下の２バイトは飛び先行番号）を中間言語バッファに出力し、ついでＢ３４Ｈ番地のルーチンをコールして文字列を数値に変換して中間言語バッファに出力し後、【処理１】の行末チェックに戻ります。このときＢ３４Ｈ番地のルーチンでＥＡＣ２Ｈ番地にに００Ｈが入ります。

【処理７】記号の処理　（８２ＥＨ～８５ＥＨ，８９５Ｈ～８ＡＥＨ）
ここでは、演算子テーブル（６Ｅ８１Ｈ～６Ｅ９５Ｈ）を用いて、文字が演算子かどうかをチェックし、演算子であればその演算子に対応する中間言語をＡレジスタに代入します。演算子でなければ、文字が００Ｈ～０８Ｈと０ＢＨ～１ＦＨのときにはその文字を２０Ｈに置換して、他の種類の文字のときにはそのまま、それぞれＡレジスタに入れます。
その後、ＥＡＢＦＨの値がＦＦＨのときにのみ、この値を００Ｈに変更して【処理５】内の、特殊コードのチェック処理に入ります。その際、Ｅ９Ｈ（中間言語の『'』）の場合以外は、戻ってきて＆定数（＆Ｈあるいは＆Ｏ）のチェックを行い、もし＆定数であれば、１４２３Ｈのルーチンをコールして、後続の、16進表記あるいは８進表記の文字列を数値に変換して中間言語バッファに出力した後、【処理１】の行末チェックに戻ります。もし、＆定数でないときには、【処理２】内の中間言語に変換をしない場合の処理に飛びます。

【処理８】変数名の処理　（８６ＣＨ～８９４Ｈ）
ここでは、【処理５】において予約語でないと判定された文字列の処理を行います。
まず、ＥＡＢＦＨにＦＦＨを入れ、次に、最初の１文字（【処理５】から来たので必ず英文字）を中間言語バッファに出力した後、英数字かピリオド『.』が続く限り、以後の文字列を行入力バッファから中間言語バッファへと転送します。転送が終わった後、【処理１】の行末チェックに戻ります。

2－3　中間言語逆変換

一度中間言語に変換したテキストは、そのままでは画面上にリストアウトすることができません。画面上に出力するためには、逆変換の処理を行って、再度、文字列のテキストに戻さなければなりません。

2－3－1　中間言語変換と逆変換

さて、０００００H～７ＦＦＦH番地のテキスト領域には、必ず中間言語に変換されたテキストが収められます。中間言語のままで収めておいたほうが、実行も速く、サイズも小さくて済むからです。ＬＩＳＴコマンドによってテキストを出力するときには、中間言語テキストを１行ずつ（１行は最大255文字)、もとのＬＩＳＴ形式のテキストに逆変換して画面に出力します。

2－3－2　中間言語逆変換処理（１９４ＣＨ）

中間言語逆変換の処理ルーチンは、ＬＩＳＴのルーチン(１８Ｄ９Ｈ～１９４ＢＨ)、ＥＤＩＴのルーチン（６５７ＢＨ～６６４１Ｈ）の２ケ所からコールされおり、引数として、ＨＬレジスタ内にウィンドウ上のテキスト行の先頭アドレスが与えられます。出力は、Ｅ９Ｂ９Ｈ番地からＥＡＢＡＨ番地までの行入力バッファに入った１行分のＬＩＳＴ形式のテキストです。

中間言語逆変換：１９４ＣＨ

領　域	１９４ＣＨ～１Ｂ３ＦＨ
機　能	中間言語形式のテキストをＬＩＳＴ形式に戻す
ワーク	ＥＡＢＥＨ：文字列の出力の制御フラグ ビット０：文字列定数（" "内の文字列）であることを表す ビット１：ＤＡＴＡ文中の文字列であることを表す ビット２：ＲＥＭ文中の文字列であることを表す ＥＣ２１Ｈ：スペース出力の制御フラグ ＥＦＢＣＨ～ＥＦＢＤＨ：中間言語テキスト内でのエラー位置を示す ＥＦＣ０Ｈ～ＥＦＣ１Ｈ：ＬＩＳＴ形式テキストでのエラー位置を示す
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＨＬ：テキストポインタ

０００００Ｈ～７ＦＦＦＨ：テキストエリア
ＥＦＢＣＨ～ＥＦＢＤＨ：テキスト上のエラー位置

出　力　ＨＬ：テキストポインタ
Ｅ９Ｂ９Ｈ～ＥＡＢＡＨ：行入力バッファ
ＥＦＣ０Ｈ～ＥＦＣ１Ｈ：行入力バッファ上のエラー位置

ルーチン　ＲＳＴ１０Ｈ：テキストからの１文字読み込み
０Ａ８ＤＨ：ＲＳＴ１０Ｈ実行後のＦＡＣの形式変換
１Ａ０ＡＨ：ＥＡＢＥＨ番地のビット１を立てる
１Ａ１４Ｈ：ＥＡＢＥＨ番地のビット２を立てる
１ＡＢ４Ｈ：（ＨＬ）から１文字読み込み、文字の種類をチェックする
２０Ｆ２Ｈ：ＨＬをインクリメントする
２２８４Ｈ：ＢＣをデクリメントする
２８Ｄ０Ｈ：ＦＡＣの数値を外部形式に変換し、ＥＣ５３Ｈ～ＥＣ６９Ｈに出力する
２ＤＢＣＨ：ＦＡＣの数値を外部形式の16進数に変換し、ＥＣ５３Ｈ～ＥＣ６９Ｈ番地に出力する
２ＤＢＦＨ：ＦＡＣの数値を外部形式の８進数に変換し、ＥＣ５３Ｈ～ＥＣ６９Ｈ番地に出力する
５２１６Ｈ：（ＨＬ）から１文字読み込み、英大文字か否かチェックする
５５８ＢＨ：スタックを変更（ＰＯＰを１回）して２つ前のルーチンに戻る
ＥＤ１ＢＨ：フック、ＲＥＴ
ＥＤＡ８Ｈ：フック、ＲＥＴ

動　作　ウィンドウ上のテキストのうち、ＨＬで示された場所から１行終了マークを表す００Ｈまでの部分を、ＬＩＳＴ形式に変換して行入力バッファに出力します。この処理は大きく４つの部分に分けることができます。

【処理Ｉ】　初期設定：Ｉ９４ＣＨ～Ｉ９５ＤＨ

レジスタとワークの初期設定を行います。

レジスタＢＣには行入力バッファの先頭アドレスＢ９Ｅ９Ｈを、レジスタＤにはリスト１行中の文字数の上限としてＦＦＨを、そしてワークのＥＡＢＥＨとＥＣ２１Ｈには００Ｈをそれぞれ設定します。このときレジスタＨＬには、上位ルーチンから引き渡された、ウィンドウ上のテキストのアドレスが入っています。

次に、ＥＤＡ８Ｈ番地のルーチンを呼び出して、Ｐオプションのプロテクトがかかっているかどうかをチェックします。その内容は、ＥＣ２９Ｈ番地（プログラムのプロテクトフラグ）が０かどうかをを調べて、０でなければＢ０６Ｈ番地へ飛んでIllegal function callエラーを出

力するというものです。
　初期設定が終わると【処理２】のエラー位置を求める部分に飛びます。

【処理２】　文字種類の割り出し：１９５ＥＨ～１９Ｄ２Ｈ
　まず、このブロックの先頭には、ＢＣ・Ｄ・ＨＬの各レジスタの更新の部分があります。ここで、Ｄレジスタが０になった場合には、文字数が超過したとみなして変換処理を終わります。ただし、このレジスタ更新の部分は、ループ制御のためのものなので、最初（【処理１】から来た場合）はスキップされます。
　次に、画面上のエラー位置を求めます。つまり、逆変換を行っているウィンドウ上の文字位置とＥＦＢＣＨ～ＥＦＢＤＨに入っているエラー位置のアドレスとを比較して、等しい場合には、ＥＦＣ０Ｈ～ＥＦＣ１ＨにＢＣレジスタの値（ウィンドウ上の文字位置に対応する、行入力バッファ上のＬＩＳＴ形式のテキストの文字位置）を代入します。この結果、ＥＦＣ０～ＥＦＣ１Ｈには、ＬＩＳＴ形式のテキスト中のエラー位置が入ることになります。
　次に、ＨＬをポインタにしてウィンドウから１文字取り出し、文字の種類を割り出します。割り出す種類とその処理は次のとおりです。

○	００Ｈ	１行の終了マーク； 変換処理を終了する
○	０１Ｈ～０ＡＨ ２０Ｈ～２１Ｈ ２３Ｈ～２ＤＨ ２ＦＨ ３ＢＨ～７ＦＨ	ノーマルなキャラクタ； そのまま行入力バッファへ１文字を出力し、ＥＣ２１Ｈ番地に００Ｈを出力する。その後、【処理２】の先頭に戻る
○	０ＢＨ～１ＦＨ ２ＥＨ ３０Ｈ～３９Ｈ	数値関係のキャラクタ； ＥＣ２１Ｈ番地の値を調べた結果がＦＦＨの場合にのみ、行入力バッファへ２０Ｈを出力する このとき行入力バッファの文字数が255文字を超過したら、変換処理を終了する。最後に、ＥＣ２１Ｈ番地に０１Ｈを出力して、【処理４】に飛ぶ
○	２２Ｈ	ダブルクォーテーションマーク『”』； ＥＡＢＥＨ番地のビット０を反転した後、行入力バッファへ文字『”』を出力し、ＥＣ２１Ｈ番地に００Ｈを出力する。その後、【処理２】の

先頭に戻る

○　３ＡＨ　　　　　コロン『：』；
ＥＡＢＥＨ番地のビット０が０である場合、ビット１をクリアする。次に、行入力バッファへ文字『：』をそのまま出力し、ＥＣ２１Ｈ番地に００Ｈを出力する。その後、【処理２】の先頭に戻る

○　８０Ｈ～ＦＦＨ　予約語；
【処理３】に飛ぶ

【処理３】　予約語の逆変換：１９Ｄ３Ｈ～１Ａ０９Ｈ、１Ａ１ＢＨ～１ＡＢ３Ｈ

ここではまず、ＥＡＢＥＨ番地の値を調べ、そのビット０からビット２までの状態により、図２－３－１のような動作をします。ここで、ＥＡＢＥＨの各ビットがセットされているときの意味についてですが、ビット０：文字列定数であること、ビット１：ＤＡＴＡ文であること、ビット２：ＲＥＭ文であることをそれぞれ示しています。

次は予約語テーブルを参照するための下準備です。

まず、テキストエリア上の予約語がＦＦＨで始まる２バイトコードのグループかどうかを調べ、２バイトコードの場合にはＦＦＨの後の１文字を読み、７ＦＨでマスクして、予約語テーブルを参照するためのコードにします。もちろん、２バイトコードでないときは、何も手を加えずにもとの１バイトを参照用コードに用います。

さらに、これらテーブル参照用のデータ１バイトが、９ＦＨ（中間言語の『ＥＬＳＥ』）もしくはＡＦＨ（同じく『ＷＨＩＬＥ』）でないかどうかを調べます。ここで９ＦＨだったときは２２８４Ｈ番地のルーチンでレジスタＢＣをデクリメントし、ＡＦＨのときはその次の１バイトがＦ３Ｈである場合のみその１バイト（Ｆ３Ｈ）を読みとばします。

以上の前処理のあと、予約語テーブル（６Ｂ８ＡＨ～６Ｅ８０Ｈ）と演算子テーブル（６Ｅ８１Ｈ～６Ｅ９５Ｈ）を用いた予約語の逆変換の処理に入ります。

ここでは、予約語テーブル中の中間言語の部分を先頭から順に、前処理で求めた参照コードと一致するまで比較を繰り返し、同時にセパレータ（予約語の頭文字が変わる部分で、００Ｈが置かれている）の数をカウントします。カウンタの初期値は４０Ｈです。

予約語テーブル中に参照コードと同じ数が見つかったらそのテーブルのアドレスをとっておいて、ついで、その参照コードによっていくつかの種類の処理を行います（図２－３－２参照）。すなわち、演算子（１文字の予約語）、通常の予約語、特殊な予約語Ｅ０Ｈ（『ＵＳＲ』）・Ｅ１Ｈ（『ＦＮ』）の３種類です。演算子かどうかは、カウントしたセパレータの値を調べればわかります。つまり、演算子の場合にはカウンタの値は５ＢＨであり、そのほかの予約語の場合にはカウンタの値は４１Ｈ（ＡＳＣＩＩコードの『Ａ』）～５９Ｈ（同、『Ｘ』）となります。

<table>
<tr><th>ＥＡＢＥＨの状態
７６５４３２１０</th><th>意　　味</th><th>動　　　作</th></tr>
<tr><td>－－－－－０００</td><td>地の文
（下記の３種類の文字列のいずれでもない、普通のプログラムの部分）</td><td>中間言語テキスト中の１文字を見て８４Ｈ(『ＤＡＴＡ』)のときはＥＡＢＥＨのビット１を立て、８ＦＨ（『ＲＥＭ』）のときは、同じくビット２を立てる。</td></tr>
<tr><td>－－－－－－－１</td><td>ダブルクォート(『"』）で囲まれた文字列の内部</td><td rowspan="2">処理２の先頭に戻る。</td></tr>
<tr><td>－－－－－０１０</td><td>ＤＡＴＡ文の文字列の内部</td></tr>
<tr><td>－－－－－１－０</td><td>ＲＥＭ文の文字列の内部</td><td>中間言語テキスト中の１文字を見たときＥ９Ｈで、かつすでに変換した文字が『：ＲＥＭ』であったとき、行入力バッファのポインタＢＣを４文字分前に戻す（つまり『：ＲＥＭ』＋Ｅ９Ｈを『’』に変換する）。</td></tr>
</table>

図２－３－１　文字列状態フラグＥＡＢＥＨの意味と動作

【処理４】数値の処理：１ＡＢＦＨ～１Ｂ３ＦＨ

ここでは、数字、小数点、数値を表すいくつかのコードに対しての処理を行います。

ＲＳＴ１０Ｈを用いて、再度テキストから１文字を読み込んだあと、Ａ８ＤＨ番地のルーチンを用いてＲＳＴ１０Ｈ用のＦＡＣ（ＥＡＣ４Ｈ～ＥＡＣＢＨ番地）から演算用のＦＡＣ（ＥＣ３ＣＨ～ＥＣ４４Ｈ番地）に書き換えます。

その後、テキスト中の数字の先頭にコード０ＢＨがあるときは２ＤＢＣＨ番地、０ＣＨがあるときは２ＤＢＦＨ番地、コードが何もついていない数字は２８Ｄ０Ｈ番地のそれぞれのルーチンに飛び、８進数、16進数、および10進数の文字列にそれぞれ変換します。ここで変換された文字列はＥＣ５３Ｈ～ＥＣ６９Ｈ番地の文字列化バッファに出力されています。

次に、数値が8進数と16進数のときには、記号『&O』および『&H』を先に出力したあと、変換した数字を文字列化バッファから行入力バッファに1文字ずつ書き写します。最後に数値が単精度か倍精度の場合、数字の最後に『！』か『#』をそれぞれ付け加え、【処理2】のエラー位置処理の部分（レジスタ更新の次の部分）に飛びます。

参照コード	処　　理
E0H，E1H	『USR』と『FN』。EC21H番地が00Hかどうかを調べ、0でなかったときにはスペースを1文字だけ行入力バッファに出力する。また、このときはEC21H番地には00Hを出力する。
演算子	EC21H番地に00Hを入れる。
その他	EC21H番地が00Hかどうか調べ、00Hでなかったときは、スペースを1文字だけ行入力バッファに出力する。EC21H番地にはFFHを出力する。

図2－3－2　予約語テーブル参照後の処理

第3章 プログラムの実行

CONTENTS

3－1　プログラム実行処理の概要

中間言語に変換されたＢＡＳＩＣのプログラム（テキスト）は、内包するコマンド・ステートメントを基本単位とし、これに関数や変数が加わって形成されます。これらの基本単位をまとめて『文』と呼ぶのです。

『文』実行の処理系は、９Ｅ５Ｈ番地の１文実行ルーチンを中心にして、９９６Ｈ番地（次の文の実行）・９Ｃ４Ｈ番地（次の行の実行）の各ルーチン、そして、各文の処理ルーチンから成り立っています。特に、ダイレクトモードのときには、４Ａ７Ｈ番地のＭＡＩＮルーチンの方から１文実行のルーチンにエントリします。

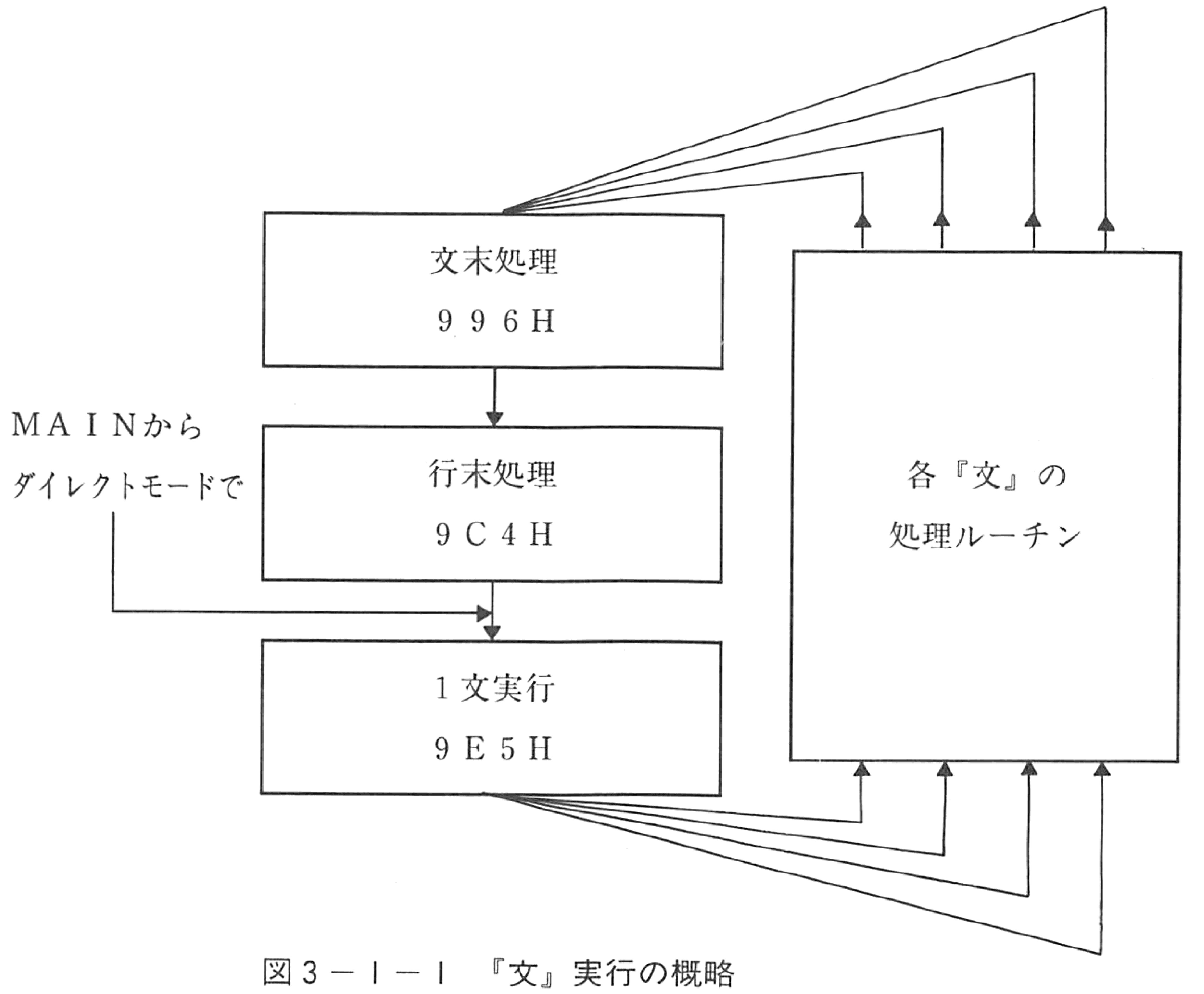


図３－１－１　『文』実行の概略

3－1－1　各文の処理ルーチン開始アドレス

各文の処理ルーチンの開始アドレスは、６９ＦＥＨ～６Ｂ５５Ｈにあるアドレステーブルを参照することによって獲得します。

例）PRINT（中間コード：91H）

６９ＦＥＨ＋(９１Ｈ－８１Ｈ)×２＝６Ａ１ＥＨ

```
MON

h]D6A1E
6A1E 54 0E 2E 52 D9 18 DD 77   T .Rル ンw
6A26 01 0D 00 18 D7 15 84 1B        ラ ■
```

――０Ｅ５４ＨがＰＲＩＮＴ文の処理ルーチン開始アドレス

☆以上のように，中間コード８１Ｈ～ＦＥＨに対応する，処理アドレステーブルが６９ＦＥＨ～６ＡＡＦＨにある。また，

中間コードＦＦ８１～ＦＦＡ９に対応するテーブルは６ＡＢ０Ｈ～６Ｂ０１Ｈに
中間コードＦＦＤ０～ＦＦＥ４に対応するテーブルは６Ｂ０２Ｈ～６Ｂ５５Ｈに

それぞれ存在する。

☆なお，ＭＩＤ＄（中間コード：ＦＦ８３）に限り，文（ステートメント）と関数の両方の形があり、テーブル上には関数用の処理アドレスしか載っていないため、ＭＩＤ＄ステートメントの処理ルーチン（５８５ＡＨ）への分岐は，テーブルを参照せずに，直接ＭＡＩＮ（９Ｅ５Ｈ）から行われる。

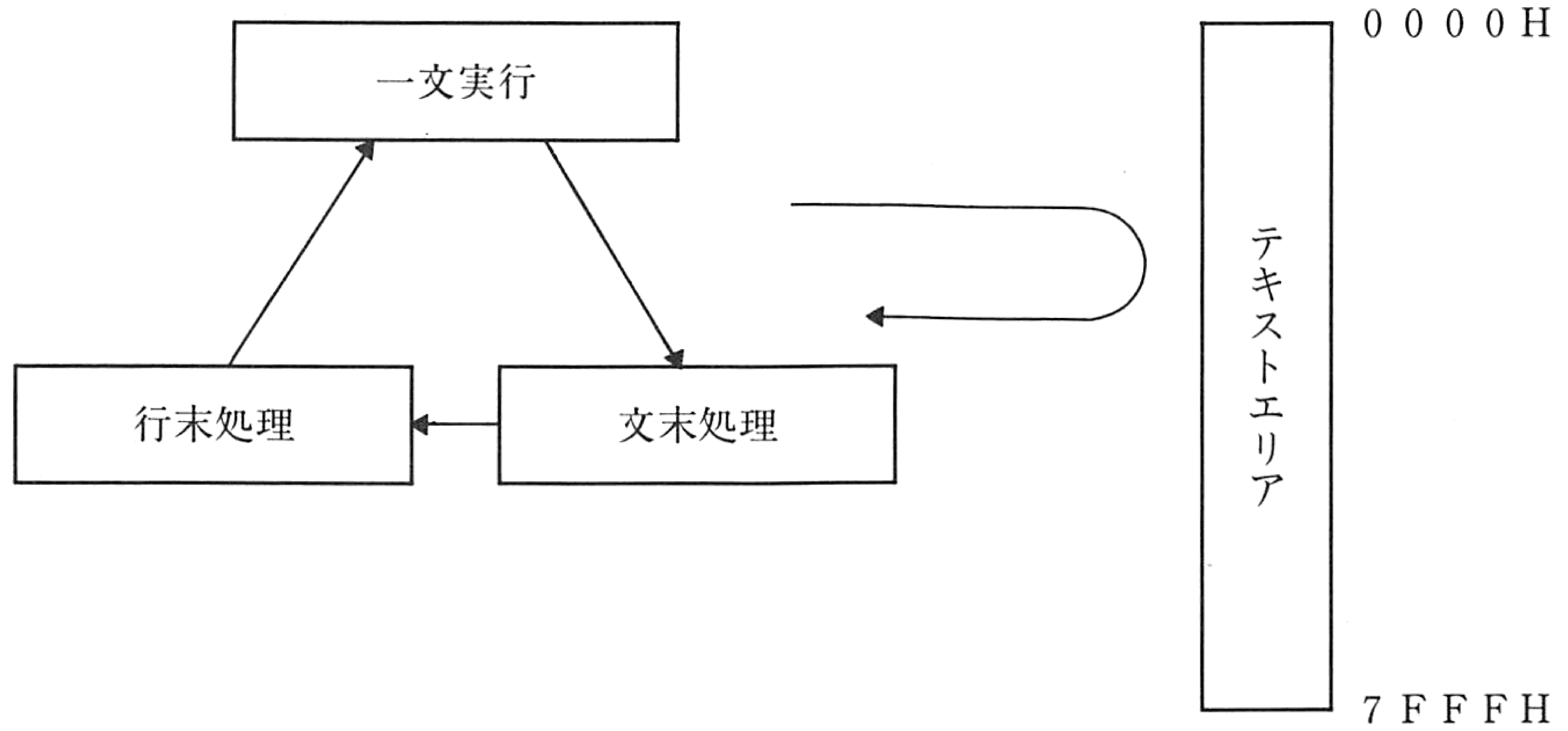
図3－1－2　中間コードの例

３－１－２　プログラム実行モードとダイレクトモード

通常、プログラム実行モードにおいては、実行時のポインタがＢＡＳＩＣテキストエリア内（００００Ｈ～７ＦＦＦＨ）を指すように設定され、このポインタが指している中間言語の取り込み・分類を基礎にして処理の実行を進めます。そして、ポインタによる処理の実行は、テキスト終了マーク（リンクポインタの値が００００Ｈになっている行）・ＥＮＤ文・ＳＴＯＰ文につき当たるまで続けられます。

これに対してダイレクトモードでは、文を中間言語に変換した後、実行時のポインタは中間言語バッファ内を指すように設定されます。

○プログラム行の実行



○ダイレクトモードでの実行

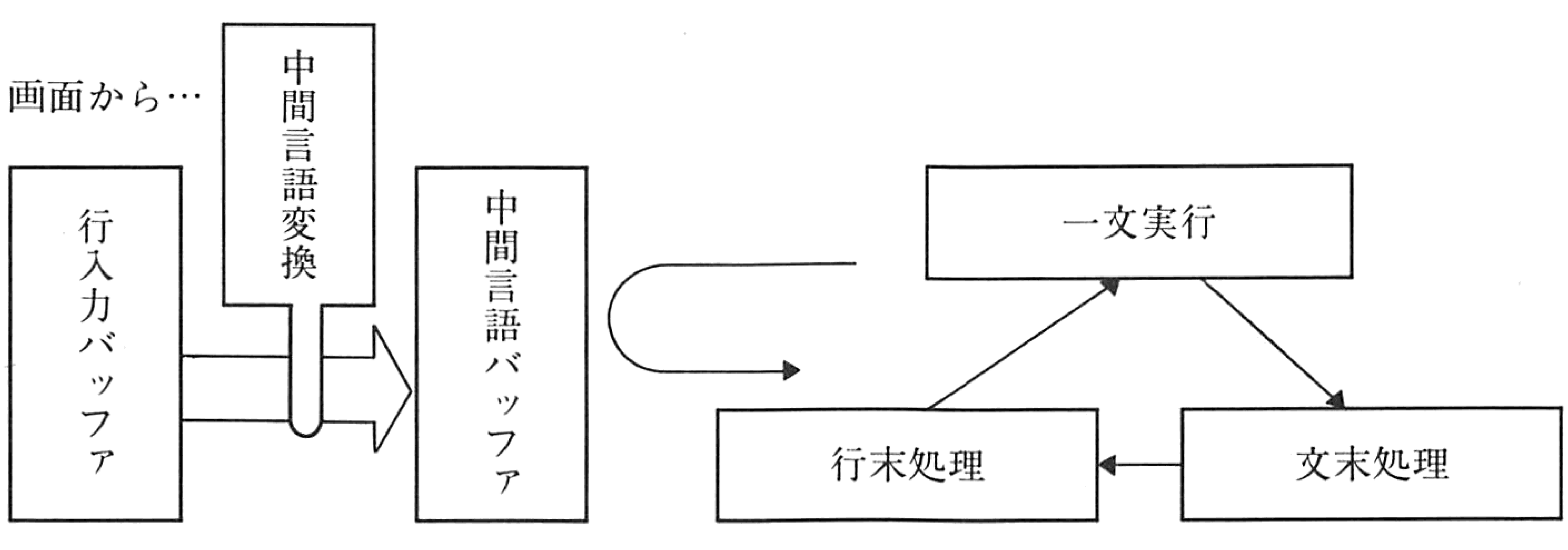


図３－１－３　プログラムモードとダイレクトモード

3－1－3　1文実行の処理

1文実行の処理は大きく分けて、3つの部分から成り立ちます。

①　1文実行前後の処理：割り込みのチェックと、『^C』や『^S』などのイベントキーのチェックを行って、1文実行前後の行番号とスタックポインタ値をそれぞれのワークに入れたりする、996Hのルーチン

②　1行実行前後の処理：テキストの終了行かどうかをチェックするほか、トレースモードのときには、行番号の表示などを行う、9C4Hのルーチン

③　1文実行の処理　　：中間言語をもとにして各文の処理の開始アドレスを割り出し、そのアドレスへ分岐して処理を実行する、9E5H番地のルーチン。

実際には、これらの3つのルーチンはひとまとまりになっているのですが、機能的に見ると別扱いにすべきであり、現実にエントリアドレスも3つ存在するので、別々のルーチンとみなすのです。それでは、3つのルーチンを説明しましょう。

1文実行：9E5H

領　域	9E5H～A0CH
機　能	各文の処理ルーチンに飛ぶ
ワーク	なし
破　壊	A，B，C，D，E，H，L
入　力	HL：テキストポインタ テキストエリア
出　力	A：『文』の2番目の文字 HL：テキストポインタ
ルーチン	0A0DH：テキストから1文字読み込む（RST10Hと同じ） 0C9CH：LET文の処理ルーチン 17C9H：中間言語2バイトグループ（FF～） 5480H：ラベルスキップ RST10H：1文字読み込み

動　作　各文の処理ルーチンのアドレステーブル（メインROM、69FEH～6B55H）を参照して、値が81H以上の中間言語から文の処理ルーチンの先頭アドレスを求め、そのアドレスに飛んで『文』を実行します。中間言語の値が80H以下だった場合には、予約語ではなく変数であるとみなしてLET文の処理に飛びます。

まず、１０Ｈ番地のルーチンで『文』の最初の１文字を読み込みます。そして、読み込んだ文字が００Ｈ（行末）か３ＡＨ（文末）のときは、４Ａ７Ｈ番地のＭＡＩＮルーチンに戻ります。また、文字がＦ５Ｈ（『＊』）のときはラベルとみなし、５４８０Ｈ番地のルーチンを用いて『＊』以後のラベルの文字列を読み飛ばします。

次に、文字の値が８０Ｈ以上かどうかを見ます。７ＦＨ以下の場合には、変数の代入文であるとみなし、ＬＥＴ文の処理ルーチン（Ｃ９ＣＨ番地）へ飛びます。

さらに、文字の値がＤ９Ｈ以上かどうかを見ます。中間言語の８１ＨからＤ８Ｈまでの部分とそれ以外のものとに分類できるのです。Ｄ９Ｈ以上（ＦＦＨも含めて）の場合には１７Ｃ９Ｈ番地に飛びます。

８１ＨからＤ８Ｈまでの場合、６９ＦＥＨ番地からのアドレステーブルのうち、６ＡＦＦＨ番地までの部分を用いてアドレスを求め、最後に次の文字を読み込んだ後、求めたアドレスに飛びます。

Ｄ９Ｈ以上の場合には、処理が少し複雑になります。

中間言語の予約語の中でＤ９Ｈ～ＦＥＨの部分と、ＦＦ８１～ＦＦＡ９の部分（ＦＦ８３を除く）は文の先頭に来ることは許されません。したがって、これらのコードが文の先頭に来た場合には、Syntax error となります。１７Ｃ９Ｈのルーチンでは、これらのコードのチェックを行った後、ＦＦＤ０～ＦＦＥ４の各予約語の処理ルーチンの先頭アドレスを求めるために、ＤＥレジスタに６Ｂ０４Ｈの値を入れて１４９９Ｈのルーチンに飛びます。

１４９９Ｈ番地では処理ルーチンのアドレステーブル中の６Ｂ０４Ｈ～６Ｂ５５Ｈの部分を参照して処理アドレスを求めたあと、そのアドレスが００００Ｈの場合には、Syntax error とします。そうでない場合には、文中の次の１文字を読み込んでおいて（１０Ｈ番地のルーチンを使用)、求めたアドレスに分岐します。

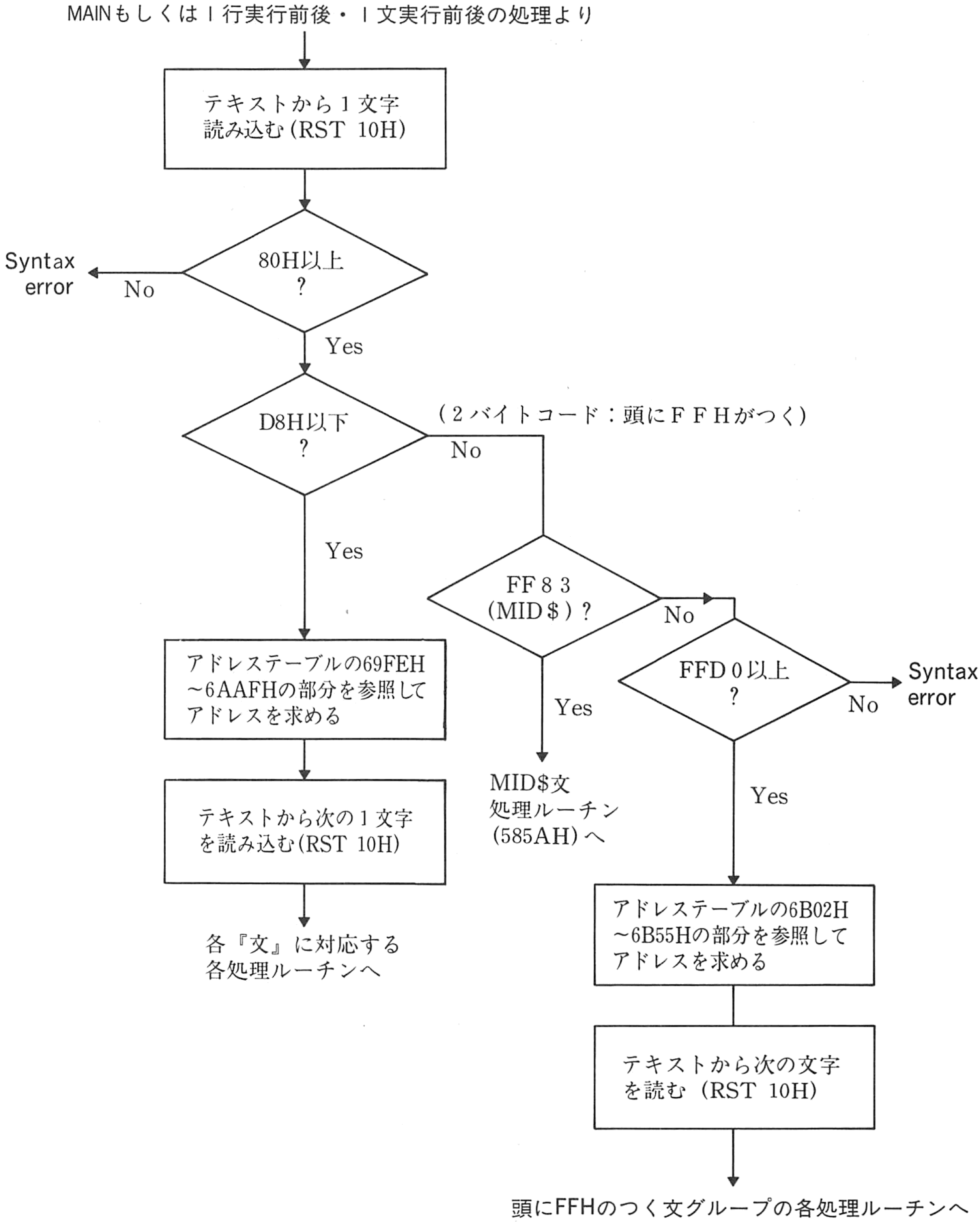


図３−Ｉ−４　Ｉ文実行処理の様子

I文実行前後の処理：９９６Ｈ

領　域	９９６Ｈ～９Ｃ３Ｈ
機　能	１文実行前後の各種チェックを行う
ワーク	なし
破　壊	Ａ，Ｆ，Ｈ，Ｌ
入　力	ＨＬ　　：テキストポインタ
	Ｅ６Ｃ９Ｈ：Comunication buffer overflow フラグ
	Ｅ６Ｆ１Ｈ：ＯＮ割り込みフラグ
出　力	ＨＬ　　：テキストポインタ
	ＥＢ０５Ｈ：１文実行前後のアドレス
	ＥＢ０７Ｈ：１文実行前後のスタックポインタ値
ルーチン	０３９３Ｈ：Syntax error 処理ルーチン
	０９Ｅ５Ｈ：１文実行の処理
	４４Ｄ５Ｈ：ウィンドウ内アドレス（ＨＬ）→　実アドレス（ＨＬ）
	５０５９Ｈ：ポーリング
	５Ａ８６Ｈ：イベントキーのチェック
	７ＣＣＤＨ：Line buffer overflow エラー処理
	ＥＤＣＣＨ：フック（ＲＥＴ）
動　作	文を実行する前後の各種のチェックを行います。

ＯＮ割り込み・イベントキーのチェック、ＥＢ０５Ｈ番地への１文実行前後のアドレス値の書き込み、ＥＢ０７Ｈ番地へのスタックポインタ値の書き込みなどを行った後、テキスト中の１文字がコロン『：』かどうかを調べます。コロンの場合には９Ｅ５Ｈ番地のルーチンに、そうでない場合には９Ｃ４Ｈ番地のルーチンに分岐します。

まず、ポート７１Ｈ番地にＦＦＨを書き込み、メモリバンクのイニシャライズを行います。

ＯＮ割り込みのチェックは、Ｅ６Ｆ１Ｈ番地（ＯＮ割り込みフラグ）を調べるだけです。ＯＮ割り込みがかかっている場合には、５０５９Ｈ番地のルーチンによって、割り込みの種類のチェックと１文実行アドレスの変更などを実行します。

イベントキーのチェックは５Ａ８６Ｈ番地のルーチンによって行われます。イベントキーとは、『^Ｃ』（ＳＴＯＰキー）や『^Ｓ』、『^Ｏ』などのキーのことです。５Ａ８６Ｈ番地のルーチンでは、イベントキーフラグ（Ｅ６ＣＡＨ）を調べて、セットされている場合には、各イベントキーに対応した処理ルーチンに飛び、そうでない場合には、何もせずにリターンします。

１文実行前後のアドレスは、現在のテキストポインタ（ＨＬレジスタ）の値を、４４Ｄ５Ｈのルーチンを用いてウィンドウ内アドレスから実アドレスに変換した後、ＥＢ０５Ｈ番地に書

き込みます。また、１文実行前後のスタックポインタ値は、現在のスタックポインタの値をそのままＥＢ０７Ｈ番地に書き込むだけです。

最後に、テキストから１文字持ってきて、『：』（３ＡＨ）かどうかをチェックします。ここで、『：』の場合には、次の９Ｃ４Ｈ番地のルーチンをスキップして９Ｅ５Ｈ番地に進みます。そうでない場合には、９Ｃ４Ｈ番地へ入ります。

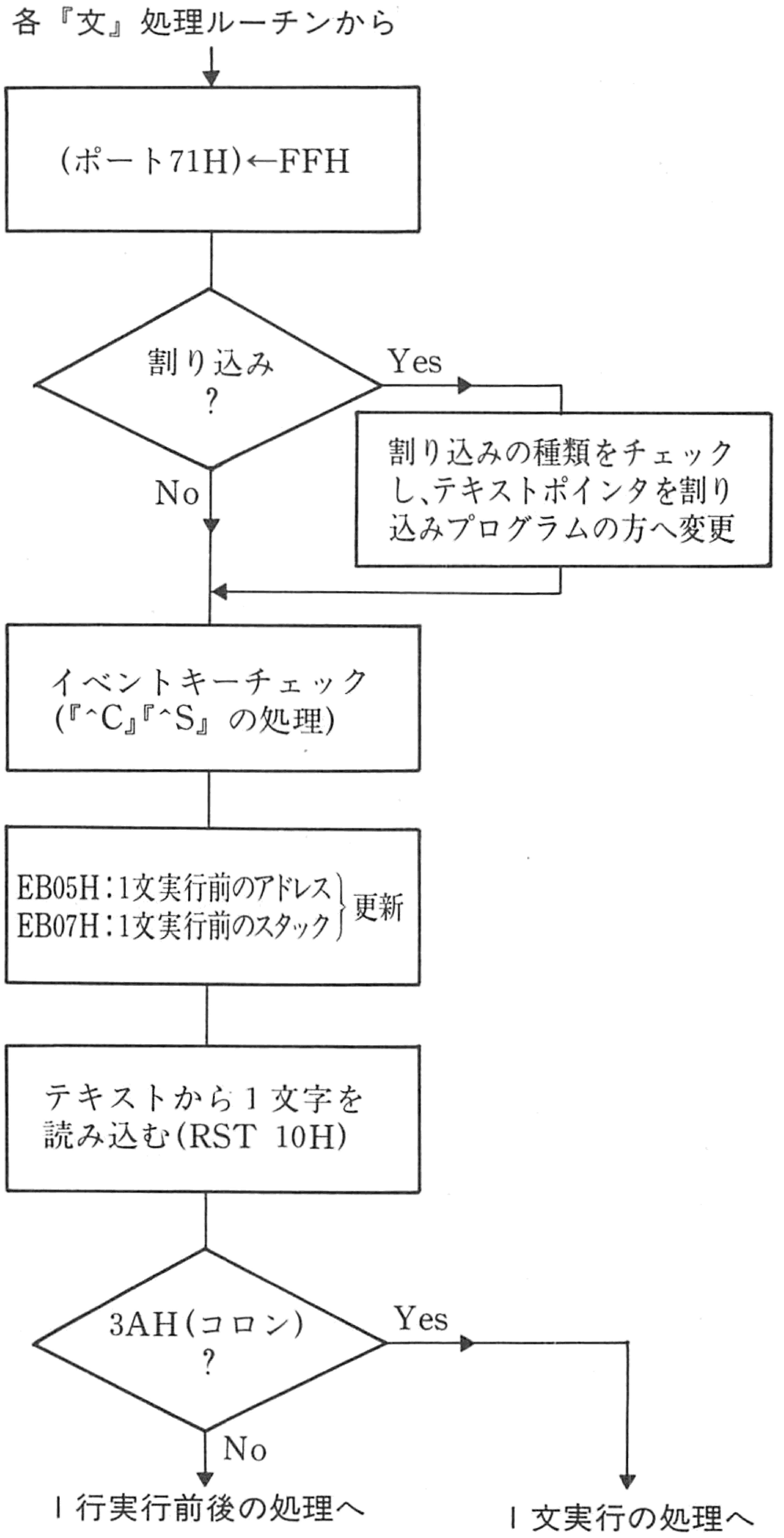


図３－１－５　１文実行前後の処理の様子

1行実行前後の処理：９Ｃ４Ｈ

領　域	9C4H～9E4H
機　能	1行実行前後の各種チェックを行う
ワーク	テキストエリア
破　壊	A，B，C，D，E，H，L
入　力	HL：テキストポインタ EC3BH：TRONフラグ
出　力	HL：テキストポインタ E656H：実行中の行番号
ルーチン	0375H：プログラムエンド処理 28C2H：HLの数値を画面に出力 44E3H：プログラムエンドチェック RST18H：画面へ1文字出力

動　作　1行分の実行が終わった後、次の行を実行する前に、テキストの終了かどうかのチェックを行って、次の行番号を捜し出します。トレースモードにおいては、その行番号の画面表示もこのルーチンで行います。

まず、テキスト終了のチェックを44E3H番地のルーチンを用いて行います。ここで行うチェックとは、行の先頭にあるリンクポインタ（2バイト）が0000Hかどうかを調べるというものです。ダイレクトモードのときにはここで必ず終わりになります。テキスト終了の場合には、375H番地に飛んでプログラムエンドの処理を行い、そこから4A7H番地のMAINルーチンに戻ります。

テキスト終了でない場合には、次にテキストからリンクポインタの次に位置する行番号2バイトを取って来てそれをE656H番地に書き込みます。E656H番地は現在実行中の行番号を示すワークエリアです。

最後に、EC3BH番地をみてトレースモードかどうかをチェックし、トレースモードだった場合には実行中の行番号（E656H番地の値と同じ）を28C2H番地のルーチンを用いて画面に出力します。

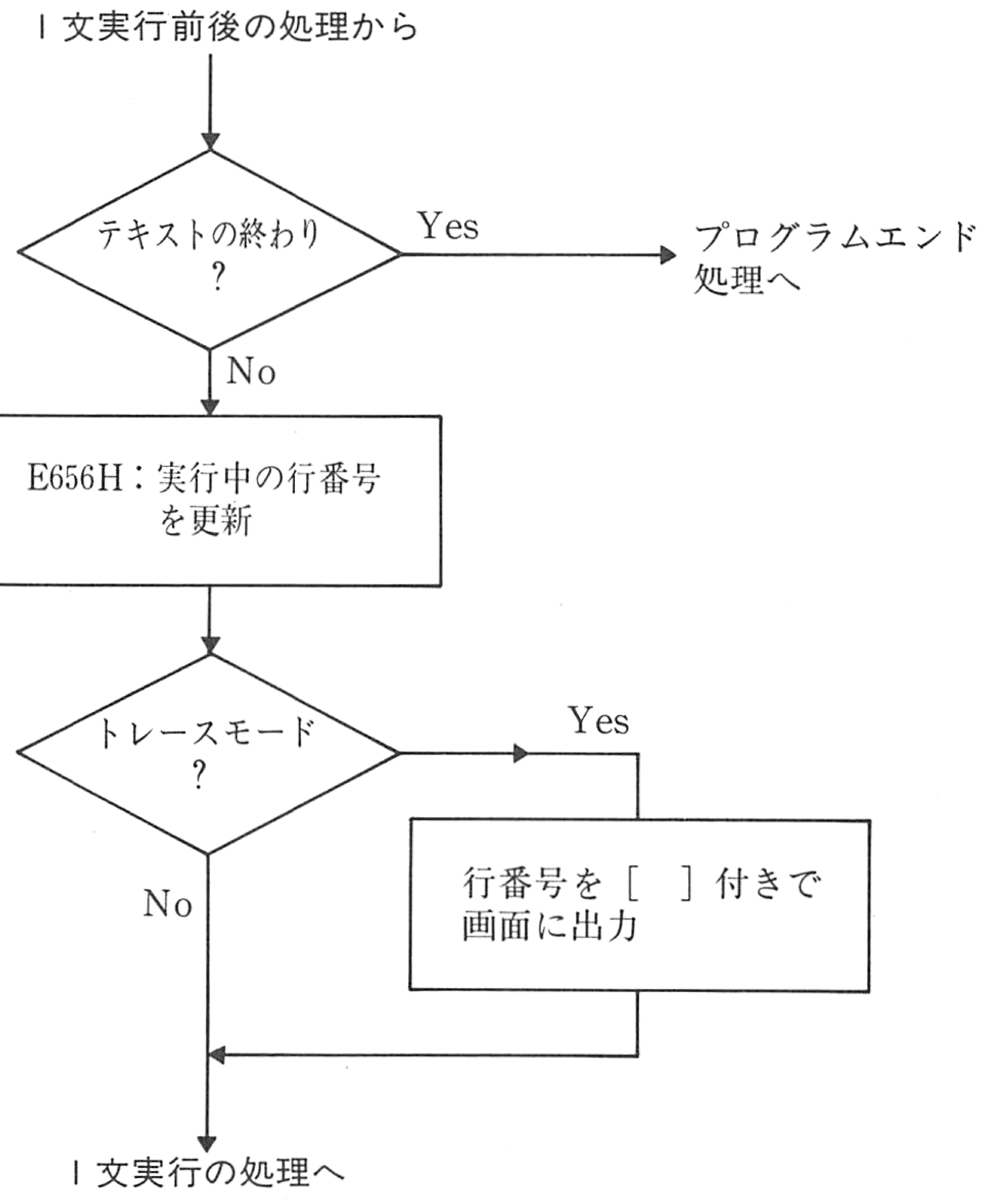


図3−I−6　I行実行前後の処理の様子

3－1－4　1文実行例

では、１文を実行する例として、プログラム実行のモードでのＩＦ文とGOTO文をとりあげてみます。なお、ＬＥＴ文については『第４章　式の評価』を参照して下さい。

例文）

```
１００　Ａ＝０：Ｂ＝０
２００　ＩＦ　Ａ＝０　ＴＨＥＮ　４００　ＥＬＳＥ　Ｂ＝０
３００　Ａ＝Ｂ
４００　Ｂ＝１：ＧＯ　ＴＯ　２００
```

上記のプログラムが実行中であるとすると、以下のような一連の処理によってプログラムの実行手順が成り立っています。

（1）　100行のＬＥＴ文（『Ｂ＝０』）実行します。

ＬＥＴ文の説明は『第４章　式の評価』を参照して下さい。
ＬＥＴ文実行後は次のような処理となります。

①　レジスタＨＬに次の文のテキストポインタが入っている状態で１文実行前後の処理（９９６Ｈ番地）に戻ります。

ここでＯＮ割り込みとイベントキーのチェック、およびＥＢ０５Ｈ番地とＥＢ０７Ｈ番地へそれぞれ１文実行前後のアドレスとスタックポインタ値の書き込み等を行います。

②　１００行の行末に来ているので１行実行前後の処理（９Ｃ４Ｈ番地）に移ります。

ここではテキストエンドのチェックと次の行番号（この場合は２００行）の割り出し、トレースモード時の行番号表示を行います。

③　１文実行の処理（９Ｅ５Ｈ番地）に移ります。

ＲＳＴ１０Ｈで読み込んだ１文字の種類で文の種類（この場合は８ＢＨ：ＩＦ文）を割り出し、さらにもう１文字ＲＳＴ１０Ｈで読み込んだ後、割り出した文のエントリに飛びます。ここでリターンアドレスとして数値９９６ＨをＰＵＳＨして置きます。

（2）　200行：ＩＦ文の処理・・・式が『真』の場合

式が『真』なので、ＧＯＴＯ文の処理ルーチンを用いて、ＴＨＥＮの後の飛び先行番号（４００行）の行のアドレスにテキストポインタを変更します。その後は（1）と同じく、１文実行前後の処理、１行実行前後の処理、１文実行の処理を続けて行います。

（3）　４００行：ＬＥＴ文（『Ｂ＝１』）の処理

このＬＥＴ文の処理を終わった後、まだ行の終わりに来ていないので、１文実行前後の処理から１文実行の処理に飛び、次のＧＯＴＯ文に移ります。

（4）　同じく４００行：ＧＯＴＯ文の処理

テキストポインタを飛び先（２００行）の行アドレスに変更した後、（1）と同じく、１文実行前後の処理、１行実行前後の処理、１文実行の処理を続けて行います。

（5）　再び２００行へ：ＩＦ文の処理・・・式が『偽』の場合

式が『偽』なので、ＥＬＳＥの後の文を実行します。すなわち、『ＴＨＥＮ』から『ＥＬＳＥ』までのテキストを読み飛ばした後、ＬＥＴ文（『Ｂ＝０』）を実行し、行末を検出したので（1）と同じく、１文実行前後の処理、１行実行前後の処理、１文実行の処理を続けて行います。

なお、ＥＬＳＥ文の処理ルーチンはＲＥＭ文のルーチンと同一の物です。これは、ＩＦ文の式が『真』の場合、ＴＨＥＮの後の文を実行後にＥＬＳＥから後の文を実行せずに次の行へ進むためです。そのため、ＥＬＳＥ文はＲＥＭ文の代わりとして使うこともできます（逆はできません）。もちろん、このような使い方はプログラムを見づらくするのでやめたほうが良いでしょう。

ＩＦ文処理ルーチン：Ｅ０５Ｈ

領　域	Ｅ０５Ｈ～Ｅ４ＢＨ
機　能	ＩＦ文の処理
ワーク	なし
破　壊	Ａ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＨＬ：テキストポインタ テキストエリア ＥＡＣ４Ｈ：ＲＳＴ１０Ｈ用のＦＡＣ
出　力	ＨＬ：テキストポインタ テキストエリア
ルーチン	０９ＥＢＨ：１文実行の処理ルーチン内 ０Ａ０ＤＨ：１文字入力（ＲＳＴ１０Ｈと同じ） ０ＢＦ９Ｈ：ＧＯＴＯ文処理ルーチン内 ０Ｃ７７Ｈ：ＤＡＴＡ文処理ルーチン １１Ｄ３Ｈ：式の評価

２０ＢＤＨ：ＦＡＣの符号を調べる
４４Ａ４Ｈ：実アドレスＨＬ　→　ウィンドウ内アドレスＨＬ
５４４１Ｈ：ラベル名からラベルの付いている行のアドレスを求める
このほか、ＲＳＴ０８Ｈ、１０Ｈ使用

動　作

（１）１１Ｄ３Ｈ番地のルーチンを用いてＩＦの後の式の評価を行います。

（２）式の直後の１文字がコンマ（『,』；２ＣＨ）だった場合には、Ａ０ＤＨ番地のルーチンを用いて式の次の１文字を読みます。

このＡ０ＤＨルーチンは、ほとんどＲＳＴ１０Ｈと同じです。異なる部分はルーチンの最初の部分にあります。Ａ０ＤＨ番地から始まるときには、最初の１文字を読んだ後、インクリメントしたテキストポインタをＥＦＢＡＨ番地に待避しますが、ＲＳＴ１０では２文字以降の文字からしか待避を行いません。

（３）　処理（２）で読んだ文字が８９Ｈ（『ＧＯＴＯ』）でもＤＤＨ（『ＴＨＥＮ』）でもないときには、Syntax error となって終わります。

文字が８９ＨかＤＤＨのときは、２０ＢＤＨ番地のルーチンを用いてＦＡＣに入っている処理（１）の結果の符号を調べ、式の結果が０（式が『偽』）のときは処理（５）に飛び、それ以外（式が『真』）は次の処理（４）に進みます。

（４）　『ＴＨＥＮ』、『ＧＯＴＯ』、『ＥＬＳＥ』、の後に続く実行文の処理を行います。

ＲＳＴ１０Ｈで『ＴＨＥＮ』または『ＥＬＳＥ』（あるいは『ＧＯＴＯ』）の次の１文字を読み、文字の種類によって以下の５種類の処理に別れます。

①　００Ｈ（行末）か３ＡＨ（文末）：１文実行前後の処理（９９６Ｈ番地）に戻って終わります。

②　Ｆ５Ｈ（中間言語の『＊』）：飛び先のラベルがあるとみなし、５４４１Ｈ番地のルーチンを用いてラベルの付いている行のアドレスを捜し、テキストポインタをそのアドレスに変更、４４Ａ４Ｈ番地のルーチンでウィンドウ内アドレスに変換した後に１文実行前後の処理へ戻って終わります。

③　０ＥＨ（中間言語で『次の２バイトは飛び先行番号』）：ＧＯＴＯ文処理ルーチン（ＢＦ９Ｈ番地）に飛びます。

④　０ＤＨ（中間言語で『次の２バイトは飛び先アドレス』）：テキストポインタをＥＡＣ４Ｈ番地（ＲＳＴ１０Ｈ用ＦＡＣ）の２バイトの値に変更し、ウィンドウ内アドレスに変換してた後、１文実行前後の処理に戻って終わります。

⑤　それ以外：１文実行の処理ルーチン内の９ＥＢＨ番地に飛んで終わります。

（５）　ＩＦ文の式の評価が『偽』だったときの処理を行います。

まず、ＤＡＴＡ文の処理ルーチン（Ｃ７７Ｈ番地）をコールしてテキストを文末まで読み飛ばします。これはテキスト中の『ＴＨＥＮ』の後の実行文の列を、『ＥＬＳＥ』が来るまで読み飛ばすためのものです。そのほか、Ｃ７７Ｈ番地のルーチンには、テキスト中の『ＩＦ』文の数をカウントしてレジスタＤに出力する機能もあります。読み飛ばした後で行末に来たときには、１文実行前後のルーチン（９９６Ｈ番地）に戻って終わります。

行末でなかったときには、ＲＳＴ１０Ｈで次の１文字を読み込みます。

文字が9ＦＨ（『ＥＬＳＥ』）であり、かつ、読み飛ばした『ＴＨＥＮ』の後の実行文の中に『ＩＦ』文がなかった場合は、処理（４）に飛んで『ＥＬＳＥ』の後の実行文の処理を行います。

それ以外の場合には、テキストの読み飛ばしを続けて行末まで進んだ後、１文実行前後の処理（９９６Ｈ番地）に戻ります。

ＧＯＴＯ文処理ルーチン：ＢＦ９Ｈ

領　域	ＢＦ９Ｈ～Ｃ４０Ｈ
機　能	ＧＯＴＯ文の処理
ワーク	なし
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＨＬ：テキストポインタ テキストエリア Ｅ６５６Ｈ：実行中の行番号 ＥＡＣ０Ｈ：ＲＳＴ１０Ｈ用　テキストポインタ ＥＡＣ２Ｈ：ＲＳＴ１０Ｈ用　読み込んだ文字
出　力	ＨＬ　　：テキストポインタ ＥＡＦＦＨ：行番号　→　行アドレスフラグ テキストエリア
ルーチン	０３９３Ｈ：Syntax error メッセージ出力 ０３Ｂ３Ｈ：エラーメッセージを出力する レジスタＥにエラーコードを入れてジャンプ ０６０５Ｈ：レジスタＤＥと同じ行番号の文を捜す ０Ｂ３４Ｈ：行番号か行アドレスを読み込んでレジスタＤＥに出力する ０Ｃ７９Ｈ：行末が来るまでテキストを読み飛ばす

１Ｃ７１Ｈ：テキスト3バイトをレジスタＡとＤＥの値に書き換え
４４Ａ４Ｈ：実アドレスＨＬ　→　ウィンドウ内アドレスＨＬ
４４Ｂ３Ｈ：ウィンドウアドレスＢＣ　→　実アドレスＢＣ
４４Ｄ５Ｈ：ウィンドウ内アドレスＨＬ　→　実アドレスＨＬ

動　作　下記（１）〜（５）のとおり。

（１）　Ｂ３４Ｈのルーチンを用いてテキストから飛び先の行番号（または行アドレス）をレジスタＤＥに読み込みます。

（２）　ＥＡＣ２Ｈ番地から、ＧＯＴＯ文の次の１文字を取り出します。ＧＯＳＵＢ文処理ルーチンやＲＥＴＵＲＮ文処理ルーチンからもここに飛んできています。

文字が０ＤＨ（中間言語で次の２バイトが行アドレス）であった場合には、レジスタＤＥに入っている飛び先行アドレスをレジスタＨＬに移した後ウィンドウアドレスに変換し、９９６Ｈ番地に戻ります。

また、文字が０ＥＨだった場合には次の処理（３）に、０ＤＨでも０ＥＨでもなかった場合には３９３Ｈ番地に飛び、Syntax error になって終わります。

（３）　Ｃ７９Ｈ番地のルーチンをコールして、次の行の先頭が来るまでテキストを読み飛ばします。これは、次の６０ＥＨ番地のルーチンによる行番号サーチのときに、テキストポインタが行の先頭になくてはならないがための処理です。

（４）　処理（１）で求めた飛び先行番号とＥ６５６Ｈ番地に入っている実行中の行番号とをＲＳＴ２０Ｈを用いて比較し、飛び先行番号が現在の行番号よりも大きいときには６０ＥＨ番地、小さいときには６０５Ｈ番地のルーチンをコールします。

この２つのルーチンは、開始アドレスが違うだけで、本体は同じ一つのルーチンです。レジスタＤＥに入っている数値と同じ行番号を持つ行のアドレスを、６０５Ｈ番地をコールしたときにはテキストの先頭から、６０ＥＨ番地をコールしたときにはレジスタＨＬに入っている現在の行番号からそれぞれテキストの終わりまで、リンクポインタを使ってサーチしています。行が見つからなかったときには、３Ｂ３Ｈ番地に飛んで、Undefined line number エラーとなって終わります。

（５）最後にＥＡＦＦＨ番地に０ＤＨ（中間コード：次の２バイトは飛び先アドレス）を書き込み、さらに１Ｃ７１Ｈ番地のルーチンを用いて、テキスト上の飛び先行番号の部分（０Ｅ，ＸＸ，ＸＸ）を飛び先アドレス（０Ｄ，ＹＹ，ＹＹ）に書き換えたあと、９９６Ｈ番地（１文実行前後の処理）に戻ります。

3－2　スタックならびに割り込みを使用する文の処理

3－1節でBASICプログラム実行の核の部分と、一般的な『文』の処理については説明が終わりました。しかしながら、「FOR～NEXT」等のループ形成文や「GOSUB」のような文ではスタックを使用する処理があり、また、「ON　割り込み　GOSUB文」では、割り込みテーブルを使った割り込み処理というように特殊な処理が行われており、これらについては、あらためて説明を加える必要があります。

3－2－1　スタックを使用する文の処理

これから説明を行うのは、FOR～NEXT、WHILE～WEND、GOSUB～RETURNの3つの処理です。これらの処理の共通点は、スタックを数バイト用いてテーブルを作成し、ループもしくはネストの処理を構成していることです。FOR～NEXTで19バイト、WHILE～WENDで7バイト、GOSUB～RETURNでは7バイト、それぞれスタックを用いています。

これらの数バイト（あるいは十数バイト）にわたるデータは、FOR文、WHILE文、もしくはGOSUB文によってスタックに積まれ、NEXT文、WEND文、RETURN文によって参照されます。以後、これら3種類のテーブルをそれぞれ、「FORテーブル」、「WHILEテーブル」、そして「GOSUBテーブル」と呼ぶことにします。

3－2－1（1）　FOR～NEXT

ループを作る文の中で、最も一般的なのがFOR～NEXTのループです。

FOR文処理：8BFH

領　域	8BFH～9E4H，52BFH～537CH（NEXTルーチン内）
機　能	FOR文の処理を行う
ワーク	EABDH：変数の型
	EAF7H：FOR文の次の文のアドレスを待避させるエリア
	EAFBH：変数の型（変数アドレスのGETで参照される）
	EAFDH：TEMP（このルーチンでは変数アドレス待避に使用）
	EC19H：FOR文の変数初期値待避（4バイト）
	EC1DH：NEXT文行番号

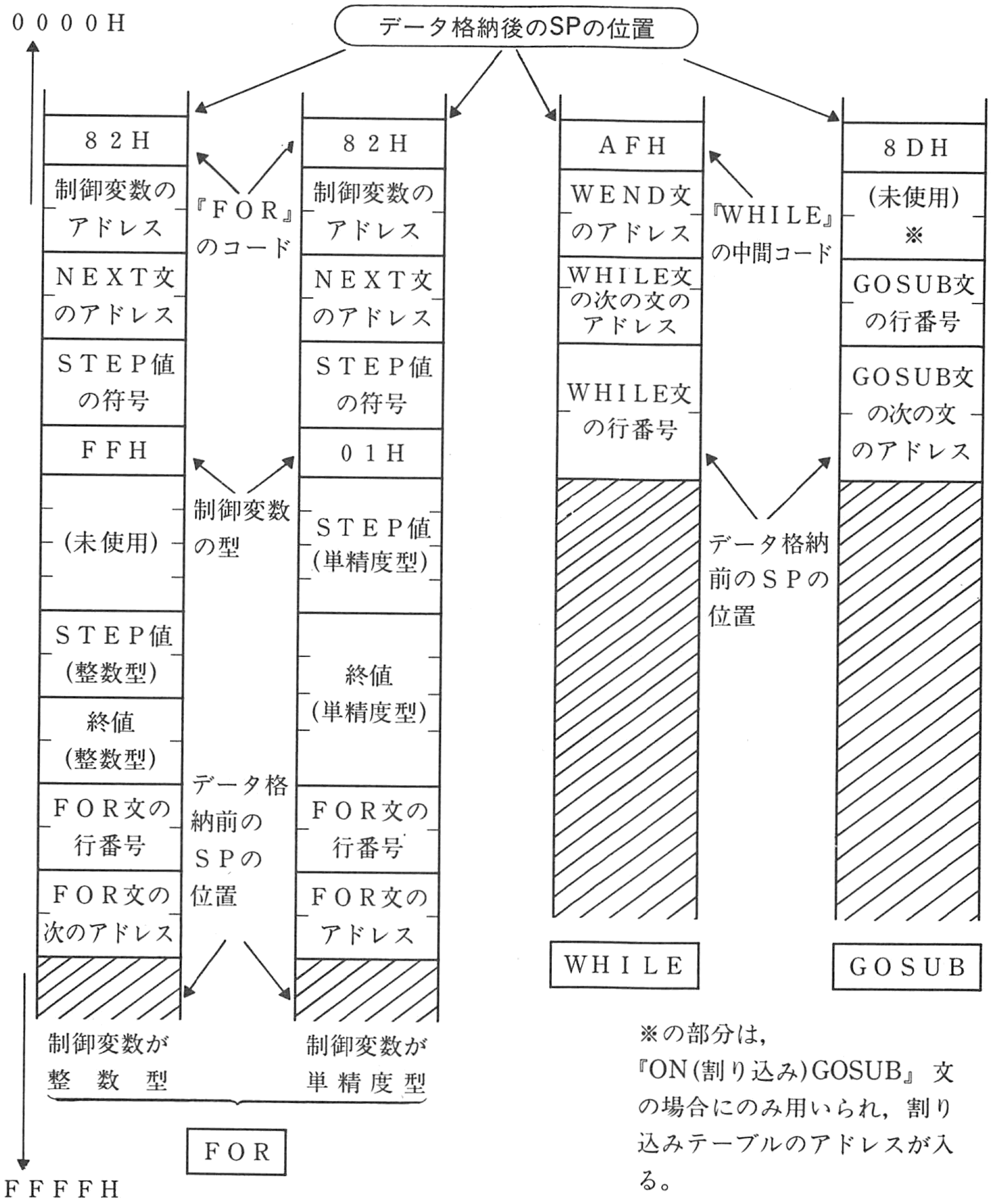


図3－2－1　スタック内テーブル

破　壊　A，F，B，C，D，E，H，L
入　力　HL　　　：テキストポインタ
テキストエリア：FOR文テキスト
E656H番地：実行中の行番号
出　力　HL　　　：テキストポインタ
EC19H：FOR文制御変数初期値（4バイト）
単純変数テーブル：制御変数のデータ
スタックエリア：FORテーブル（19バイト）
ルーチン　034AH：スタック内サーチ
0C77H：DATAエントリ
11D3H：式の評価
1796H：FAC型変換
1895H：整数式の評価
1D28H：ループエンドサーチ
20C7H：HLの値によってAレジスタとフラグをセットする
20E8H：単精度型FAC　→　BCDEレジスタ
20F4H：単精度型FAC　→　(HL)～(HL＋3)
21A0H：FACをINT型に変換する
2214H：FACを単精度に変換
44D5H：ウィンドウ内アドレスHL　→　実アドレスHL
4EC1H：Out of memory チェック
5ACAH：変数アドレスGET
そのほか、18H・38Hを除く、RSTルーチン全種類使用
動　作　次の手順で処理しています。

①　5ACAH番地のルーチンを用いて変数アドレスを求めます。ここで制御変数に配列を使った場合にはエラーとなります。求めた変数アドレスは、EAFDH番地に待避しておき、後でスタックに積み込みます。HLレジスタ（テキストポインタに使用）は変数名の次の場所を指します。

②　RST08Hを用いて変数名の次に『＝』が来ているかどうかをチェックします。

③　11D3H番地のルーチンを用いて、変数の初期値の評価を行います。結果はFACに出力されます。

④　1796H番地のルーチンを用いてFACの内容をEABDH番地に示されている変数型に変換します。変換したFACの内容は、単精度型の4バイトだけをEC19H～EC1CH番地に待避します。この4バイトは後のNEXT文のルーチン内で使用されます。

⑤　テキストポインタを待避しておいてDATAエントリ（C77H番地）をコールして、コ

ロン『：』か行の終わりが来るまでテキストを読み飛ばし、ＦＯＲ文の次の文のアドレスを求めます。求めたアドレスはＥＡＦ７Ｈ番地に待避して置き、後でスタックに積み込みます。
⑥　３４ＡＨ番地のルーチンを用いてスタック内をサーチし、①で求めた変数アドレスと一致する変数アドレスがスタック内のＦＯＲテーブルに存在するかどうかを調べます（サーチは、ＦＦＦＦＨ番地の方に向かって、ＦＯＲおよびＷＨＩＬＥのテーブルがなくなるまで続けられます）。該当するアドレスが存在しない場合には何も変化ありませんが、もしも存在する場合には、さらに、ＦＯＲ文の次の文のアドレスについて、処理⑤でＥＡＦ７Ｈ番地に待避しておいた２バイトとＦＯＲテーブル中の２バイトとを比較して、一致しなければサーチを繰り返す一方、一致した場合、ＳＰレジスタとＥＢ０７Ｈ番地（１文実行前後のスタック）の内容を見つけたＦＯＲ文のレベルへの変更を行います。つまり、ＦＯＲループやＷＨＩＬＥループが何重にもネストしている中からＧＯＴＯ文等で一番外側のＦＯＲ文に飛んだ場合には、その内側にあるループはすべて白紙に戻されてしまうことを意味します（もちろん、内側にあってもすでに終了しているループには関係ありません）。
⑦　４ＥＣ１Ｈ番地のルーチンを用いてスタックがスタックエリアからはみ出さないかどうかのチェックを行います。はみ出す場合には、Out of memory エラーです。
⑧　スタックへの積み込みを始めます。まず処理５で求めたＦＯＲ文の次の文のアドレス、次にＥ６５６Ｈ番地（実行中の行番号）の内容をスタックに積み込みます。スタックへの積み込みはレジスタのＰＵＳＨ命令と「ＥＸ　（ＳＰ），ＨＬ」命令とによって行います。
⑨　ＲＳＴ０８Ｈによって初期値の後に『ＴＯ』のコードがあるかどうかをチェックしてから、ＲＳＴ３０Ｈで変数の型を調べ、倍精度型と文字列型の場合には、Type mismatch エラーとします。
⑩　終値（ＴＯの値）を積み込みます。１１Ｄ３Ｈ番地のルーチンで式の評価を行い、変数の型にＦＡＣを変換した後、単精度型のときには４バイト、整数型のときには２バイトをスタックに積み込みます。
⑪　ＳＴＥＰの値をスタックに積み込みます（テキスト中にＳＴＥＰの指定があるときには、そのＳＴＥＰ値を、指定がないときにはＳＴＥＰ値として「１」を積み込みます）。その方法ですが、変数が整数型の場合には、１８９５Ｈ番地のルーチンを用いてＤＥレジスタの２バイトに、単精度型の場合には１１Ｄ４Ｈ番地・２２１４Ｈ番地・２０Ｅ８Ｈ番地のルーチンを用いてＢＣレジスタとＤＥレジスタの４バイトに、ＳＴＥＰ値を格納した後、スタック内の終値の上に積み込みます。
⑫　整数型のときは２０Ｃ７Ｈ番地のルーチンを、単精度型のときにはＲＳＴ２８Ｈを用いることにより、ＳＴＥＰ値の符号をＣレジスタに格納し、さらにＲＳＴ３０Ｈを用いてＦＡＣの型をＢレジスタに格納して、両者をスタックに積み込みます。
⑬　ＲＳＴ１０Ｈによってテキストの次の１文字を読み込み、行の終わりか文の終わりでない場合には、Syntax error とします。

⑭　１Ｄ２８Ｈ番地のルーチンを用いて、対応するＮＥＸＴ文のアドレスを求めてスタックに積み込み、同時に求めたＮＥＸＴ文の行番号をＥ６５６Ｈ番地（実行中の行番号）に格納します。

⑮　処理①でＥＡＦＤＨに待避しておいた変数アドレスと、ＦＯＲ文の中間コード『８２Ｈ』とを順番にスタックに積み込み、ＳＰを１回インクリメントした後さらに「ＰＵＳＨ　ＡＦ」を２回繰り返します。

⑯　最後に、テキストポインタをＨＬレジスタに持たせて、ＮＥＸＴルーチン内の５２ＢＦＨ番地に飛びます。ここから先は、ＮＥＸＴ文の説明を参照して下さい。

ＮＥＸＴ文処理：５２ＢＤＨ

領　域	５２ＢＤＨ～５３７１Ｈ
機　能	ＮＥＸＴ文の処理を行う
ワーク	ＥＡＢＤＨ：ＦＡＣの型
	ＥＣ１６Ｈ：ＮＥＸＴ文アドレス
	ＥＣ１８Ｈ：ＮＥＸＴ文フラグ
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＨＬ　　　：テキストポインタ
	ＥＣ１９Ｈ：ＦＯＲ文制御変数初期値（４バイト）
	テキストエリア：ＮＥＸＴ文テキスト
	スタックエリア：ＦＯＲテーブル（19バイト）
	単純変数テーブル：ＦＯＲ文制御変数
出　力	ＨＬ　　　：テキストポインタ
	Ｅ６５６Ｈ：実行中の行番号
	ＥＢ０７Ｈ：１文実行前のスタック
ルーチン	０３４６Ｈ：スタック内サーチ
	１ＤＤＥＨ：単精度型ＦＡＣインクリメント
	２０ＤＡＨ：（ＨＬ）～(ＨＬ＋３）　→　単精度型ＦＡＣ
	２０ＥＢＨ：（ＨＬ）～(ＨＬ＋３）　→　Ｅ，Ｄ，Ｃ，Ｂ
	２０Ｆ４Ｈ：単精度型ＦＡＣ　→　（ＨＬ）～(ＨＬ＋３)
	２１３４Ｈ：ＢＣＤＥ－ＦＡＣを行ってＡレジスタ，フラグをセット
	２１５ＦＨ：ＤＥ－ＨＬを行ってＡレジスタ，フラグをセット
	２３３ＡＨ：ＤＥ＋ＨＬ　→　ＦＡＣ，ＨＬ
	４４Ａ４Ｈ：実アドレスＨＬ　→　ウィンドウ内アドレスＨＬ
	４４Ｄ５Ｈ：ウィンドウ内アドレスＨＬ　→　実アドレスＨＬ

５ＡＣＡＨ：変数アドレスＧＥＴ
そのほか、ＲＳＴ１０Ｈ、２０Ｈ使用

動　作　以下のように処理を進めます。

①　ＮＥＸＴ文用フラグをセットします。ＥＣ１８Ｈ番地のＮＥＸＴ文処理としてコールされた場合には数値ＡＦＨが、ＦＯＲ文からひきついでいる場合には００Ｈが入ります。ＮＥＸＴ文かどうかを示すフラグです。

②　テキストエリア中のＮＥＸＴ文のアドレス（テキストポインタＨＬを、４４Ｄ５Ｈ番地のルーチンによって実アドレスに変換したもの）をＥＣ１６Ｈ番地に待避しておきます（処理④で使用）。

③　ＮＥＸＴの次に変数名が続く場合に限り、５ＡＣＡＨ番地のルーチンを用いて変数テーブルをサーチします。見つかった変数のアドレスはＤＥレジスタに出力されます。ＮＥＸＴの次に変数名が続かない場合には、ＤＥレジスタには００００Ｈが入ります。また、テキストポインタＨＬは、テキスト中の変数名の次の位置アドレス（変数名がなければ『ＮＥＸＴ』の次の番地）が入り、これがＥＡＦＤＨ番地に待避されます。

④　スタックの中で対応するＦＯＲ文を捜します。３４６Ｈ番地のルーチンを用いて、スタック内に同じ変数アドレスを持つＦＯＲ文のテーブルが存在するかどうかを調べます（ＦＯＲとＷＨＩＬＥのテーブルがなくなるまで、ＦＦＦＦＨの方に向かってサーチを行います）。このとき、ＮＥＸＴの次に変数名が続いていない場合には、変数アドレスの比較は行わず、ただ、ＦＯＲテーブルを捜すだけです。発見されたときには、ＳＰレジスタの値をＦＯＲテーブルの先頭アドレス（『ＦＯＲ』のコードすなわち８２Ｈのあるアドレス）に変更する一方、発見できない場合には、Syntax error とします。さらに、見つかったスタック上のＦＯＲテーブルから、ＮＥＸＴ文アドレスを取り出し、この値と処理②でＥＣ１６Ｈ番地に待避しておいたアドレスとを比較して、一致しないときには、やはりSyntax error とします。

⑤　ＦＯＲテーブルから、ＳＴＥＰ値とその符号、制御変数の型などを取り出し、変数の値の更新を行います。

制御変数が単精度型のときには、２０ＤＡＨ番地のルーチンを用いて変数テーブルからＦＡＣに数値を読み込み、この値と単精度型ＳＴＥＰ値とを１ＤＤＥＨ番地のルーチンで加算して結果をＦＡＣに格納します。さらに、２０ＥＢＨ番地のルーチンを用いてＦＡＣの内容を変数テーブルに格納し、２１３４Ｈ番地のルーチンを用いてＦＡＣの内容をＢＣ・ＤＥの各レジスタに代入します。

制御変数が整数型のときには、サブルーチンを用いずに直接、変数の値とＳＴＥＰ値とをレジスタに格納し、２３３ＡＨのルーチンを用いて整数同士の加算を行います。結果はＦＡＣとＤＥレジスタに出力されるので、ＤＥレジスタから変数テーブルに格納します。このとき、結果が整数の範囲を超えた場合、すなわちＥＡＢＤＨ番地の値が４（単精度型）になった場合には、Over flow エラーとします。

ＦＯＲ文からのエントリの場合（ＥＣ１８Ｈ番地の値が０の場合）には、これらの変数の加算を行わず、ＥＣ１９Ｈ番地から初期値を取り出して変数テーブルに書き込みます。このとき、変数の値は他の２つの場合と同様に、単精度型がＢＣ・ＤＥの各レジスタ、整数型がＤＥレジスタに、それぞれ格納されています。

⑥　２１５ＦＨ番地のルーチンを用いて更新した変数値と変数のもとの値を比較し、さらに、その結果の符号とＳＴＥＰ値の符号とを比較します。符号が一致しない場合にはＦＯＲループの未終了を意味するので、ＦＯＲテーブルから取り出したＦＯＲ文の次の行番号をＥ６５６Ｈ番地に代入し、テキストポインタをＦＯＲテーブルから取り出したＦＯＲ文の次のアドレスに変更して９９２Ｈ番地に飛びます。９９２Ｈ番地では『ＦＯＲ』コード（８２Ｈ）をスタックに積み込み、そのまま９９６Ｈ番地のルーチン（次の文の実行）に移行します。

⑦　処理⑥で符号が一致した場合には、このＦＯＲループが終了したものとみなし、ＳＰレジスタとＥＢ０７Ｈ番地の値をＦＯＲテーブル作成前の値に戻して、ループ１段分のＦＯＲテーブルを破棄します。そして、処理③で待避したテキストポインタをＥＡＦＤＨ番地から読み出し、制御変数名（変数名がなければ『ＮＥＸＴ』）の次の１文字を読み、コンマ『，』でなければ９９６Ｈ番地に飛んで終了します。コンマだった場合には、ＲＳＴ１０Ｈによって次の文字を読み込み、５２Ｃ７Ｈ番地のルーチンをコールします。５２Ｃ７Ｈというのは、実はＮＥＸＴルーチン自身です。つまり、変数名が『，』で連結されている限り、自分自身を再起的に呼び続け、それぞれの変数について独立にＮＥＸＴ処理を行うわけです。ＮＥＸＴ文を続けて書いた場合とほぼ等価ですが、変数名が必ず付いていなくてはなりません。面白いことに、コールした自分自身には決して戻らず、コール先で必ず９９２Ｈ番地か９９６Ｈ番地に飛ぶことによってＮＥＸＴ文処理を抜けます。したがって、「ＣＡＬＬ　５２Ｃ７Ｈ」命令がこのルーチンの終端になっています。

３―２―１（２）　ＷＨＩＬＥ～ＷＥＮＤ

ＷＨＩＬＥ文処理：Ａ８７５Ｈ

領　域	Ａ８７５Ｈ～Ａ８９ＥＨ
機　能	ＷＨＩＬＥ文の処理を行う
ワーク	ＥＡＦ７Ｈ：テキストポインタの待避
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＨＬ：テキストポインタ Ｅ６５６Ｈ：実行中の行番号
出　力	ＨＬ　　　：テキストポインタ

ＥＢ０７Ｈ：１文実行前のスタック
スタックエリア：ＷＨＩＬＥテーブル（７バイト）

ルーチン　１Ｄ２ＡＨ：ループエンドサーチ
４４Ａ４Ｈ：実アドレスＨＬ　→　ウィンドウ内アドレスＨＬ
４４Ｄ５Ｈ：ウィンドウ内アドレスＨＬ　→　実アドレスＨＬ
Ａ８９ＦＨ：ＷＥＮＤ文処理ルーチン

動　作　処理の手順は次のとおり。

① ＷＨＩＬＥ文のアドレスを実アドレスに変換してＥＡＦ７Ｈ番地に待避しておきます。

② １Ｄ２ＡＨ番地のテキストエリア中の対応しているＷＥＮＤ文をサーチし、その実アドレスを求めます。

③ スタック内に同一のＷＥＮＤ文アドレスを持つＷＨＩＬＥループ用テーブルがないか、現在のＳＰの値からＦＦＦＦＨ番地に向けてＷＨＩＬＥとＦＯＲのテーブルがなくなるまでサーチを行います。見つかったときには、ＳＰを変更してそのテーブルをあらためて使用し、そのテーブルのもとの情報は破棄されます。変更したＳＰの値はＥＢ０７Ｈ番地（１文実行前後のスタック）にも格納されます。

④ 最後に、ＳＰの値、処理①でＥＡＦ７Ｈ番地に待避しておいたＷＨＩＬＥ文のアドレス、ＷＥＮＤの次のアドレス、の３つを順番にスタックに積み込み、ＷＥＮＤルーチン内のＡ８Ｃ９Ｈ番地に飛びます。ここから先はＷＥＮＤルーチンの説明を参照して下さい。

ＷＥＮＤ文処理：Ａ８９ＦＨ

領　域　Ａ８９ＦＨ～Ａ９１５Ｈ

機　能　ＷＥＮＤ文の処理を行う

ワーク　ＥＣ１ＤＨ：行番号待避

破　壊　Ａ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ

入　力　ＨＬ　　　：テキストポインタ
スタックエリア：ＷＨＩＬＥテーブル
Ｅ６５６Ｈ：実行中の行番号
ＥＢ０７Ｈ：１文実行前のスタック

出　力　ＨＬ　　　：テキストポインタ
Ｅ６５６Ｈ：実行中の行番号

ルーチン　０３９３Ｈ：Syntax error の処理ルーチン
１１Ｄ３Ｈ：式の評価
２０ＢＤＨ：ＦＡＣの符号を調べる
４４Ａ４Ｈ：実アドレスＨＬ　→　ウィンドウ内アドレスＨＬ

４４Ｄ５Ｈ：ウィンドウ内アドレスＨＬ　→　実アドレスＨＬ

動　作　処理の手順は次のとおり。

①　１文実行のルーチンですでに読み込んである文字の種類から、ＷＥＮＤが文末かどうか(ＷＥＮＤの次にある文字が００Ｈもしくは３ＡＨかどうか）をチェックし、文末でなかった場合、Syntax error となります。

②　ＷＥＮＤアドレスを実アドレスに変更後、スタック内に同一のＷＥＮＤ文アドレスを持つＷＨＩＬＥループ用テーブルがないか、現在のＳＰの値からＦＦＦＦＨ番地に向けてＷＨＩＬＥとＦＯＲのテーブルがなくなるまでサーチを行います。見つかったときにはＳＰの値ををそのテーブルの先頭番地に変更し、変更したＳＰの値はＥＢ０７Ｈ番地(１文実行前のスタック）にも格納されます。また、見つからなかったときには、ＷＥＮＤ without WHILE エラーになります。

③　Ｅ６５６Ｈ番地（実行中の行番号）から現在のＷＥＮＤ文の行番号を読み込んでそれをＥＣ１ＢＨ番地に待避しておき、レジスタＤＥにＷＨＩＬＥの次のアドレス、レジスタＨＬとＥ６５６Ｈ番地にＷＨＩＬＥの行番号をそれぞれ順にスタックから取り込みます。

④　ＷＨＩＬＥ文の式の評価を行います。１１Ｄ３Ｈ番地のルーチンを用いてＷＨＩＬＥ文の式の評価を行った後、２０ＢＤＨ番地のルーチンを用いて、ＦＡＣ上にある結果の符号を調べ、結果が０（式が『偽』）だった場合には次の処理⑤に飛び、それ以外（式が『真』）の場合は中間言語のＡＦＨ（『ＷＨＩＬＥ』）１バイトをスタックに積み込んだ後、１文実行後の処理ルーチン（９９６Ｈ番地）に戻って終わります。

⑤　ＷＨＩＬＥ文の式が『偽』だった場合の処理です。

処理③でＥＣ１ＤＨ番地に待避しておいたＷＥＮＤ文のアドレスをＥ６５６Ｈ番地に入れ、スタックからＷＥＮＤ文のアドレスを取り出すとともにＷＨＩＬＥテーブルを元に戻した（ＰＯＰを３回）後、１文実行後の処理ルーチン（９９６Ｈ番地）に戻って終わります。

３―２―１（３）　ＧＯＳＵＢ～ＲＥＴＵＲＮ

ＧＯＳＵＢ～ＲＥＴＵＲＮはループではありませんが、やはりスタック上に専用のテーブルを設けてその処理を行います。

ＧＯＳＵＢ文にはＯＮ～ＧＯＳＵＢ等の分派が多数存在するため、ＲＥＴＵＲＮ文はそれらすべてのＧＯＳＵＢ文に対応するように作られており、ＧＯＳＵＢテーブルにもそのための２バイトのデータ（割り込みテーブルのアドレス）が付加されています。

ＧＯＳＵＢ文処理：ＢＢＦＨ

領　域	ＢＢＦＨ～ＢＤＦＨ，ＢＦＣＨ～Ｃ４０Ｈ（ＧＯＴＯ文処理ルーチン内）
機　能	ＧＯＳＵＢ文の処理を行う
ワーク	なし
破　壊	Ａ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＨＬ　　：テキストポインタ Ｅ６５６Ｈ：実行中の行番号
出　力	ＤＥ　　：飛び先の行アドレス ＨＬ　　：テキストポインタ スタックエリア：ＧＯＳＵＢテーブル
ルーチン	０Ｂ３４Ｈ：行番号か行アドレスを読み込み、ＤＥに入れる ０ＢＦＣＨ：ＧＯＴＯ文処理ルーチン内 ４４Ｄ５Ｈ：ウィンドウ内アドレスＨＬ　→　実アドレスＨＬ ４ＥＣ１Ｈ：Out of memory チェック （Ｃレジスタにスタックレベルを入れてＣＡＬＬする）
動　作	以下のように処理を進めます。

①　４ＥＣ１Ｈ番地のルーチンを用いて、あと３レベル（＝６バイト）スタックに積み込んだときにスタックが文字列エリアにまではみださないかどうかをチェックします。はみ出すときには、Out of memory エラーとなります。

②　Ｂ３４Ｈ番地のルーチンを用いて、テキストから飛び先の行アドレスを（飛び先が行番号指定のままである場合には、そのアドレスを行アドレスに変換して）、ＤＥレジスタに読み込みます。

③　ＧＯＳＵＢ文のアドレスと行番号、ダミーアドレス定数ＤＤ６Ｈ（２バイト）、『ＧＯＳＵＢ』のコード：８ＤＨ（１バイト）、９９６Ｈ（２バイト）の合計９バイトをスタックに積み込みます。このうち、最後の定数９９６Ｈは、１文実行後の処理ルーチンへ戻るためのリターンアドレスで、ＧＯＳＵＢ処理のために使われるスタックは残りの７バイトだけです。このあとＢＦＣＨ番地（ＧＯＴＯ文処理ルーチン内）に飛びます。ＤＥレジスタには、飛び先となる行のアドレス、ＨＬレジスタには現在のテキストポインタが入っています。ここから先は、ＧＯＴＯ文処理ルーチンの説明を参照して下さい。

ＲＥＴＵＲＮ文処理：Ｃ４１Ｈ

領　域	Ｃ４１Ｈ～Ｃ７６Ｈ
機　能	ＲＥＴＵＲＮ文の処理を行う
ワーク	ＥＡＦＤＨ：テキストポインタの待避
破　壊	Ａ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＨＬ　　　：テキストポインタ スタックエリア：ＧＯＳＵＢ文処理用テーブル
出　力	ＨＬ　　　：テキストポインタ Ｅ６５６Ｈ：実行中の行番号 ＥＢ０７Ｈ：１文実行前のスタック
ルーチン	０３４６Ｈ：スタック内サーチ ０Ｃ７７Ｈ：ＤＡＴＡ文処理ルーチン ４４Ａ４Ｈ：実アドレスＨＬ　→　ウィンドウ内アドレスＨＬ ５００８Ｈ：エラートラップ終了時の割り込みテーブル、および、ＯＮ割り込みカウンタの操作 ＥＤ４ＢＨ：フック（ＲＥＴ） そのほか、ＲＳＴ１０Ｈを使用
動　作	以下のとおり。

①　テキストポインタをＥＡＦＤＨ番地に待避したあと、３４６Ｈ番地のルーチンを用いてスタック内のサーチを行います。３４６Ｈルーチンは、本来、ＮＥＸＴ文に対応するＦＯＲテーブルを捜すために使われます（ＮＥＸＴ文処理ルーチン参照）が、ＲＥＴＵＲＮ文では、このルーチンは、ＦＯＲとＷＨＩＬＥのテーブルの終わりのアドレスを見つけるためだけに用いられています。したがって、３４６Ｈ番地のルーチンからはＦＯＲとＷＨＩＬＥのテーブルの終わりのアドレスと、そのアドレスにあるデータ１バイトがそれぞれ出力されます。

②　ＳＰとＥＢ０７Ｈ番地の内容を３４６Ｈ番地のルーチンで求めたアドレスに変更します。これによって、サブルーチンの中で動いていたＷＨＩＬＥやＦＯＲのループはすべてキャンセルされます。

また、３４６Ｈ番地から出力された１バイトが８ＤＨ（『ＧＯＳＵＢ』の中間コード）であるかどうかをチェックし、８ＤＨでない場合には、RETURN without GOSUB エラーとします。

③「ＯＮ　割り込み　ＧＯＳＵＢ」では割り込みテーブルのアドレス、ただのＧＯＳＵＢではダミーアドレス定数ＤＤ６Ｈをスタックから取り出し、このアドレスにあるデータ１バイトのビット０がセットされているかどうかをチェックします。

セットされている場合、５００８Ｈ番地のルーチンを用いて割り込みテーブルの操作を行う一方、クリアされている場合には、何もせずに処理④に移ります。

５００８Ｈ番地のルーチンでは、割り込みテーブル中のＳＴＯＰフラグをクリアし、それまでＳＴＯＰフラグがセットされていた場合には、さらにリクエストフラグがセットされているかどうかをチェックし、セットされていたら、Ｅ６Ｆ１Ｈ番地のＯＮ割り込みカウンタをインクリメントします。

④　ＧＯＳＵＢ文の行番号とアドレスをスタックから取り出した後、処理①でＥＡＦＤＨ番地に待避しておいたＲＥＴＵＲＮ文のアドレスを取り出して、そのアドレスの次の１文字を読み込みます。

文字が文末だったときには、Ｅ６５６Ｈ番地にＧＯＳＵＢ文の行番号を入れ、ＧＯＳＵＢ文のアドレスを実アドレスでＨＬレジスタに入れてＤＡＴＡ文処理のルーチンに移ります。ＤＡＴＡ文処理ルーチンでは、ＧＯＳＵＢ文を読み飛ばした後、１文実行前後の処理（９９６Ｈ番地）のルーチンに戻ります。

文字が文末でなかったときには、ＢＦ９Ｈ番地（ＧＯＴＯ文処理のルーチン内）に分岐して、『ＲＥＴＵＲＮ　[行番号]』の処理を行います。詳しくはＧＯＴＯ文処理ルーチンの説明を参照して下さい。

３―２―２　割り込みを使う文の処理

ここでいう割り込みとは、ＣＰＵがサポートしているハード的な信号のことではなく、ソフトウェアの上で疑似的に処理される、一連の例外処理の文に関するものを指します。具体的にいうと、『ＯＮ 割り込み ＧＯＳＵＢ文』（以下「ＯＮ割り込み文」と略記）の周辺の処理のことです。

これらの割り込みは以下の３つの処理から成り立っています。

①　ＯＮ割り込み文の処理　　　：　ＯＮ割り込みの種類に対応する飛び先行番号を設定します。

②　ＯＮ割り込み許可の文の処理：　『ＨＥＬＰ　ＯＮ／ＯＦＦ／ＳＴＯＰ』等が該当し、割り込みの許可／禁止／停止の処理を行います。

③　割り込みチェックの処理　　：　これは１文実行前後の処理のルーチン内で行われ、割り込みがかかったかどうかを調べます。
割り込みがかかった場合、割り込みの種類を調べ、割り込み文で設定した飛び先行番号に飛びます。

3－2－2（1）　割り込みテーブル

割り込み文は、大きく分けて6種類あり、細かく分けると、『ON　KEY』が10通り、『ON　COM』も3通りに分けることができ、合わせて17通りの割り込みが存在することになります。

これらの割り込み文の飛び先となる行番号や割り込み許可に関するデータは3バイトで一組にまとめられ、EECBH番地から始まる合計63バイトのON割り込みテーブルに収納されます。

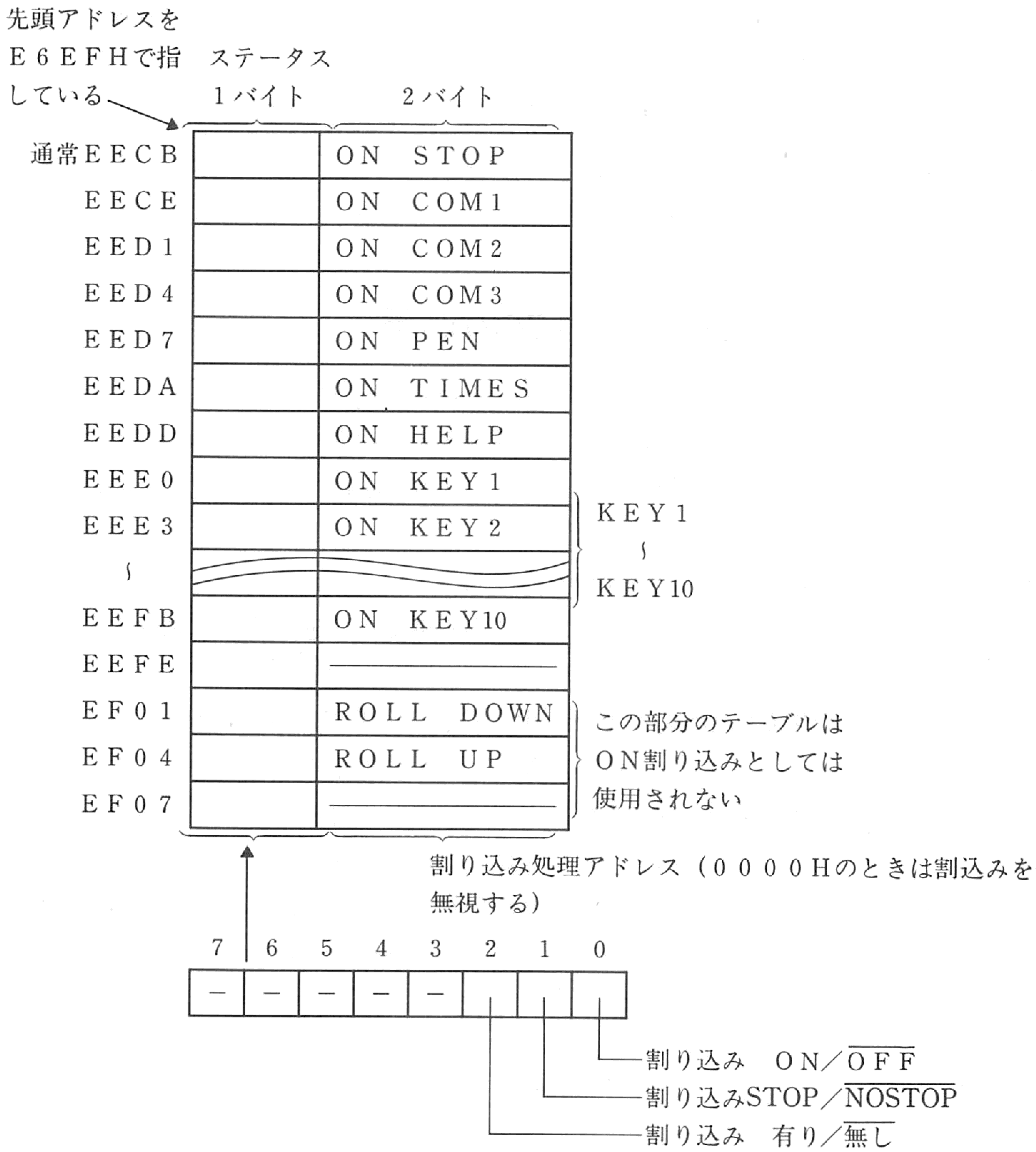


図3－2－2　割り込みテーブル

３－２－２（２）　ＯＮ割り込みで使用する処理ルーチン

『ＯＮ』文の処理ルーチンはＤ０１Ｈ番地から始まります。まずここで、ＯＮ文の種類を判別して、『ＯＮ』の次の文字がＡ５Ｈ（中間言語の『ＥＲＲＯＲ』）でない場合には、すべて続く「ＯＮ割り込み文」の処理ルーチンに飛びます。

ＯＮ　割り込み　ＧＯＳＵＢ文処理：Ｄ３４Ｈ

領　域	Ｄ３４Ｈ～Ｄ７２Ｈ
機　能	ＯＮ {ＣＯＭ, ＨＥＬＰ, ＫＥＹ, ＰＥＮ, ＳＴＯＰ, ＴＩＭＥ＄} ＧＯＳＵＢ文　の処理を行う
ワーク	なし
破　壊	Ａ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＨＬ　　：テキストポインタ テキストエリア
出　力	ＨＬ　　：テキストポインタ ＥＥＣＢＨ：割り込みジャンプテーブル（３バイト×21組） スタックエリア：ＧＯＳＵＢテーブル
ルーチン	０３９３Ｈ：Syntax error 処理ルーチン ０６０３Ｈ：行番号サーチ（→０６０５Ｈ） ０９ＥＢＨ：１文実行処理ルーチン内 ０Ｂ０ＢＨ：行番号読み込み ０Ｃ３ＣＨ：Undefined line number エラー処理ルーチン ４４Ａ４Ｈ：実アドレスＨＬ　→　ウィンドウ内アドレスＨＬ ４４Ｂ３Ｈ：ウィンドウ内アドレスＢＣ　→　実アドレスＢＣ ４４Ｄ５Ｈ：ウィンドウ内アドレスＨＬ　→　実アドレスＨＬ ６ＥＥＥＨ：４thＲＯＭ内ＯＮ～ＧＯＳＵＢサブルーチン（その１） 　　メインＲＯＭ内４４５１Ｈ番地のルーチンを通じて４thＲＯＭ内７２Ｄ０Ｈ番地のルーチンをコールする ６ＥＦ６Ｈ：４thＲＯＭ内ＯＮ～ＧＯＳＵＢサブルーチン（その２） 　　（その１）と同様にして、７３２４Ｈルーチンをコール

動　作　次のような手順で処理を行います

①　4thROMの7324H番地のルーチンをコールして、『ON』の次の文字から割り込みの種類を割り出し、同時に、Bレジスタに割り込みテーブルの番号を、Cレジスタには飛び先行番号の個数の上限（『ON KEY』が10，『ON COM』が3，他は1）を、それぞれ代入します。ここで、6種類のON割り込みのどれでもない場合（キャリーフラグがセットされた場合）には、『ON 〈式〉 GOSUB（GOTO）文』と判断し、D73H番地からの、ON 〈式〉 GOSUB（GOTO）処理ルーチンに分岐します。

②　RST10Hでブランクを読み飛ばし、割り込みの種類を表す中間コード(『COM』、『PEN』、『HELP』等）の次の文字が、『GOSUB』であるかどうかのシンタックスチェックをRST08Hで行います。

また、処理③から処理⑥まではループになっているので、そのループの回数を数えるためのループカウンタを初期値0に設定します。このループは、『ON　KEY　GOSUB文』のときなどに複数の飛び先行番号を受け付けるためのもので、1回目のループ（ループカウンタの値＝0）には最初の行番号が、また、2回目・3回目ループには2番目・3番目の行番号がそれぞれ対応しています。

③　ここから処理⑥までがループを構成しています。

まず、処理1で求めた飛び先行番号の個数の上限値と、ループカウンタの値とを比較し、カウンタが上限値を超えている場合は393H番地に飛んで、Syntax error となります。

次に、B0BH番地のルーチンを用いて飛び先の行番号を読み込みDEレジスタに入れます。飛び先が0行である場合には処理⑤に飛びます。

④　603H番地のルーチンを用いて飛び先となる行を捜し、発見された場合、その行のアドレスはDEレジスタに入れます。603Hルーチンでは、最初に、リターンアドレスをスタックの1段深いところに移し、スタックのもとの場所にはHLレジスタのテキストポインタを書き込みます。そこからあとは605H番地のルーチンと同一です。

飛び先となる行が発見されなかった場合には、C3CH番地に飛んでUndefined line numberエラーとなります。

⑤　割り込みテーブルに飛び先となる行のアドレスを書き込みます。Aレジスタに処理①で求めた割り込みテーブル番号を、DEレジスタには飛び先の行のアドレスを入れて、4thROM内の72D0H番地のルーチンをコールします。72D0Hルーチンでは、E6EFH番地に入っている割り込みテーブルの先頭アドレスと、割り込みテーブル番号を3倍したものとを加え合わせて目的の割り込みテーブルのアドレスを求め、テーブル3バイトのうちの、後の2バイトに飛び先の行のアドレスを書き込んでいます。

⑥　RST10Hでブランクを読み飛ばした後、RST08Hにより次の文字がコンマ『,』かどうかをチェックして、コンマでない場合には、373H番地に飛んでSyntax error とします。コンマの場合には、ループカウンタをインクリメントして処理③に戻ります。

割り込みチェック処理：５０５９Ｈ

領　域	５０５９Ｈ～５０Ａ４Ｈ
機　能	ＯＮ割り込みのチェックを行う
ワーク	なし
破　壊	Ａ，Ｆ　　　　　　（割り込みを行わないとき） Ａ，Ｆ，Ｂ，Ｄ，Ｅ　（割り込みを行うとき）
入　力	Ｅ６５６Ｈ：実行中の行番号 Ｅ６ＥＥＨ：ＯＮ割り込み全部の個数 Ｅ６ＥＦＨ：ＯＮ割り込みテーブルアドレス ＥＢ０ＦＨ：エラートラップルーチン実行中フラグ
出　力	なし　　　　　　　　（割り込みを行わないとき） ＢＣ　：ＯＮ割り込みテーブルのアドレス ＤＥ　：飛び先行番号 ＨＬ　：テキストポインタ （ＢＣ，ＤＥ，ＨＬ：割り込みを行うとき） ＯＮ割り込みテーブル
ルーチン	０ＢＥ０Ｈ：割り込みＧＯＳＵＢ処理 ４ＥＣ１Ｈ：Out of memory チェック ４ＦＦＢＨ：ＯＮインタラプトＳＴＯＰ ５０２ＦＨ：ＯＮ割り込みテーブル中、リクエストフラグがセットされているときの、ＯＮカウンタ（Ｅ６Ｆ１Ｈ）のデクリメント
動　作	以下のとおり。

①　ＥＢ０ＦＨ番地のエラートラップルーチン実行中フラグを調べ、０でない（エラートラップのルーチン実行中である）ときはそのまま何もせずにリターンします。

また、Ｅ６５６Ｈ番地に入っている、現在実行中の行番号を調べて、それがＦＦＦＦＨであるときもそのままリターンします。

②　ＯＮ割り込みテーブル中で「ＯＮリクエストフラグ」がセットされているものをサーチします。Ｅ６ＥＦＨ番地に入っているＯＮ割り込みテーブルのアドレスから始めて、テーブルの番号順に、Ｅ６ＥＥＨ内のＯＮ割り込みの個数分だけ、ＯＮ割り込みリクエストのチェックを繰り返し、リクエストフラグがセットされているものを発見した時点で、次の処理③に移ります。発見できない場合にはそのままリターンします。

細かくいうと、「ＯＮ割り込みリクエスト」のチェックの部分は、３バイトのテーブル中の最初の１バイトを調べて、第１ビット（ＯＮフラグ）がセット、第２ビット（ＯＦＦフラグ）がクリア、かつ、第３ビット（リクエストフラグ）がセットされた状態のときに処理③へ進むというものです。

③　ＯＮ割り込みテーブルをサーチして、割り込みリクエストがセットされているブロックから、飛び先となる行のアドレスを読み、アドレスが００００Ｈだったときには処理②の途中に戻って、リクエストフラグのサーチの続けます。
④　５０２ＦＨ番地のルーチンを用いて、Ｅ６Ｆ１Ｈ番地の「ＯＮカウンタ」をデクリメントし、さらに、４ＦＦＢＨ番地のルーチンを用いて割り込みテーブル中のＳＴＯＰフラグをセットします。
⑤　ＧＯＳＵＢテーブルを作成するのに備えるため、４ＥＣ１Ｈ番地のルーチンを用いることにより、スタックエリアに3レベル分の余裕があるかどうかをチェックし、ＢＣレジスタに割り込みテーブルのアドレス、ＤＥレジスタに飛び先行番号、ＨＬレジスタにテキストポインタを、それぞれ代入した上で、割り込みＧＯＳＵＢ処理のルーチンに飛びます。

割り込みＧＯＳＵＢ処理：ＢＥ０Ｈ

領　域	０ＢＥ０Ｈ～０ＢＦ５Ｈ
機　能	ＯＮ割り込み時のスタック処理を行う
ワーク	なし
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＢＣ　　　：割り込みテーブルのアドレス
	ＤＥ　　　：飛び先行番号
	ＨＬ　　　：テキストポインタ
	Ｅ６５６Ｈ：実行中の行番号
出　力	ＨＬ　　　：テキストポインタ
	スタックエリア：ＧＯＳＵＢテーブル
	ＥＢ０５Ｈ：１文実行前後のテキストのアドレス
ルーチン	０９Ｃ４Ｈ：１行実行前後の処理
	４４Ａ４Ｈ：実アドレスＨＬ　→　ウィンドウ内アドレスＨＬ
	４４Ｄ５Ｈ：ウィンドウ内アドレスＨＬ　→　実アドレスＨＬ

動　作　ＨＬレジスタで入力された現在実行中の次の文のテキストポインタ、Ｅ６５６Ｈ番地に入っている現在行の行番号、ＢＣレジスタで与えられた割り込みテーブルのアドレス、さらに、中間言語の８ＤＨ（『ＧＯＳＵＢ』）を、上記の順にスタックに積み込みます。最後に、ＤＥレジスタで与えられた飛び先となる行のアドレスをＥＢ０５Ｈ番地（１文実行前後のアドレス）に書き込んでから、１行実行前後の処理（９Ｃ４Ｈ番地）に分岐します。

3－3　BASICのプログラムをスピードアップする方法

ＢＡＳＩＣは、使いやすく初心者の方にもなじみやすいプログラム言語です。その反面、実行が遅いとか、大きなプログラムが作りにくいなどという欠点も、ＢＡＳＩＣを使用している人の実感だと思います。

ここでは、ＢＡＳＩＣのプログラムを少しでも速くしようと考えている人のために小手先のテクニックあれこれを紹介します。

なお、本当にスピードアップを望んでいる方には、ＢＡＳＩＣなどさっさとやめて別のプログラミング言語（「ＦＯＲＴＲＡＮ」や「Ｃ」、アセンブラ等）に転向する事をおすすめします。

その1）マルチステートメントを多用する。プログラム１行の254文字分を目一杯に利用して多数のステートメントを詰め込む。

その2）コメントはできることなら使わずに済ます。使うときには『'』ではなく『ＲＥＭ』でコメントを始めること。

その3）ラベルも使用しない。

その4）変数は、許される限り小さい型を指定し、その変数が使用される頻度の高い順に変数テーブルに並べるためにプログラムの先頭でそれぞれＤＥＦ文、ＤＩＭ文等を用いて宣言しておく。また、変数名はなるべく短いものにする。

例）
```
10 DEFINT I, J
20 DIM I, J, X, Y, Z, HAIRETSU (100)
   頻度     高い ←                    → 低い
```

その5）画面に表示する必要がなければ画面表示用のＤＭＡ転送をキャンセルする。

これは、ＳＲのハイスピードモードＢＡＳＩＣの場合には必要のないテクニックです。具体的にはポート５１Ｈ番地に数値００Ｈを出力すれば画面表示がなくなります。このあと、画面表示を再開するにはＣＲＴＣの初期設定をやりなおさねばなりませんが、これはＢＡＳＩＣのＷＩＤＴＨ命令を実行するだけで十分です。

その6）キー入力が必要ないときには、キー入力用のタイマー割り込みを禁止する。

ポートＥ６Ｈ番地のビット２をＨＩＧＨにするとタイマー割り込みが禁止されます。ポート

Ｅ６Ｈ番地は書き込み専用なので、ワークエリアのＥＦ０ＥＨ番地に保存してあるデータに手を加えてポートＥ６Ｈ番地に出力します。キー入力を再開するときにはＥＦ０ＥＨ番地のデータをもう一度ポートＥ６Ｈ番地に出力します。

これらをＢＡＳＩＣでやると以下のようになります。

```
１００　ＲＥＭ　キー入力の禁止。
２００　ＯＵＴ　＆ＨＥ６，ＰＥＥＫ（＆ＨＥＦ０Ｅ）　ＡＮＤ　＆ＨＦＤ
３００　ＲＥＭ　ユーザのプログラムが始まります。
　：　　　　　　　　　：
７００　ＲＥＭ　ユーザのプログラムの終わり。
８００　ＲＥＭ　キー入力の許可。
９００　ＯＵＴ　＆ＨＥ６，ＰＥＥＫ（＆ＨＥＦ０Ｅ）
```

これらのテクニックは本当に小手先だけのものですが、それでも合わせて使えば60％程度のスピードアップにはなります。

以上実行スピードを速くするテクニックあれこれを紹介しましたが、これらはいずれもプログラムを読みにくくしたり、バグのを誘発する原因であったりします。

そのような訳で、これらのテクニックを用いて有効なのはすでにプログラムが完成しておりなおかつそのスピードに不満を持っているときのみであるといえます。それ以外の場合には実行スピードよりも開発スピードの方が重要であり、きれいに書くテクニックに専念した方が有効です。テクニックは上手に使い分けて下さい。

第4章

式評価系

4

CONTENTS

4－1　数値データの形式

4－1－1　変数エリア

アドレス８４００H以降のメモリの配置は図４－１－１のようになっています。ここで、ディスクコードというのはディスクバージョンにおけるＢＡＳＩＣの拡張部分です。ドライブポインタからファイルバッファの部分については第５章で扱います。また、ラベルテーブルというのは、実行時にラベルが集められる領域のことを指します。本章で扱う式の評価に関係するのは、単純変数テーブル・配列変数テーブル・文字列エリア・フリースペースの４つです。これらの領域は、変数の登録やＤＩＭ文・ＥＲＡＳＥ文・ＣＬＥＡＲ文の実行、文字列の操作、ガベージコレクションなどによって、位置がずれたり大きさが変化したりします。また、スタックエリアの大きさとＴＯＰＭＥＭの値は、ＣＬＥＡＲ文で変えることができます。ユーザがマシン語エリアを使用する場合には、ＣＬＥＡＲ文でＴＯＰＭＥＭを設定することにより、マシン語用の領域を確保する必要があります。

4－1－2　単純変数の格納形式

単純変数のメモリ内での格納形式は、図４－１－２のようになっています。先頭の１バイトは識別子と呼ばれるもので変数の型を表しますが、同時に、データ部の長さをも表しています。次の変数名部というのは、変数の名前を入れておくところで少し複雑な構造となっています。初めの２バイトは変数名の先頭２文字がそのまま入ります（変数名が１文字のときの２バイト目は００Hとなります）。次のバイトには変数名の長さから２を引いた値が入ります(変数名が１文字のときは００H)。この値と識別子の値を足したものがこれ以降の変数の長さとなりますので、変数サーチにおいて関係のない変数をスキップするのに役立ちます。もしも、変数名の３文字目以降がある場合には、この後に格納されます。ただし、これらの文字はビット７が"１"の状態になっています。なお、変数名部には変数の型を表す『！』、『#』、『＄』、『%』といった記号は付けられません。最後のデータ部には、その変数の持つ値が納められます。ＶＡＲＰＴＲ関数によって得られる変数のアドレスというのは、このデータ部の先頭番地のことです。このデータ部の数値表現はＦＡＣと同じですので、ＦＡＣの項（４－１－５節）もあわせて参照してください。


8400H
ディスクコード
DRVTAB(EC81H)
ドライブポインタ
FILTAB (EC7FH)
ファイルポインタ
ドライブテーブル
ファイルバッファ
LBLTAB (EB16H)
ラベルテーブル
VARTAB (EB1BH)
単純変数テーブル
ARYTAB (EB1DH)
配列変数テーブル
STREND (EB1FH)
フリースペース
FRETOP (EAF1H)
文字列エリア
MEMSIZ (EACCH)
スタックエリア
TOPMEM (E654H)
ユーザ用マシン語エリア
E600H
MAXMEM (E7E8H)
ワークエリア
FFFFH

図4－1－1　メモリマップ

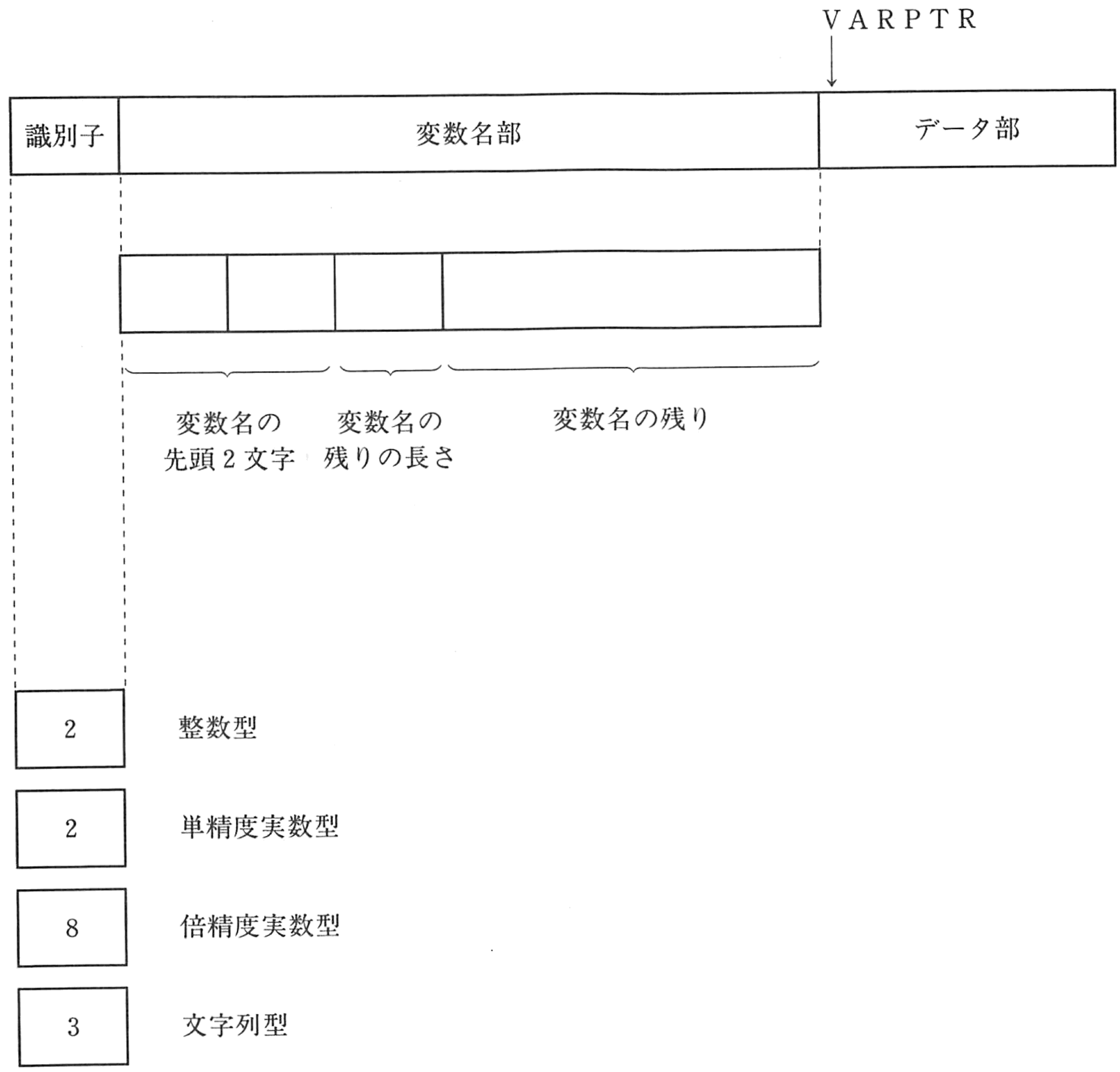


図4－1－2　単純変数の構造

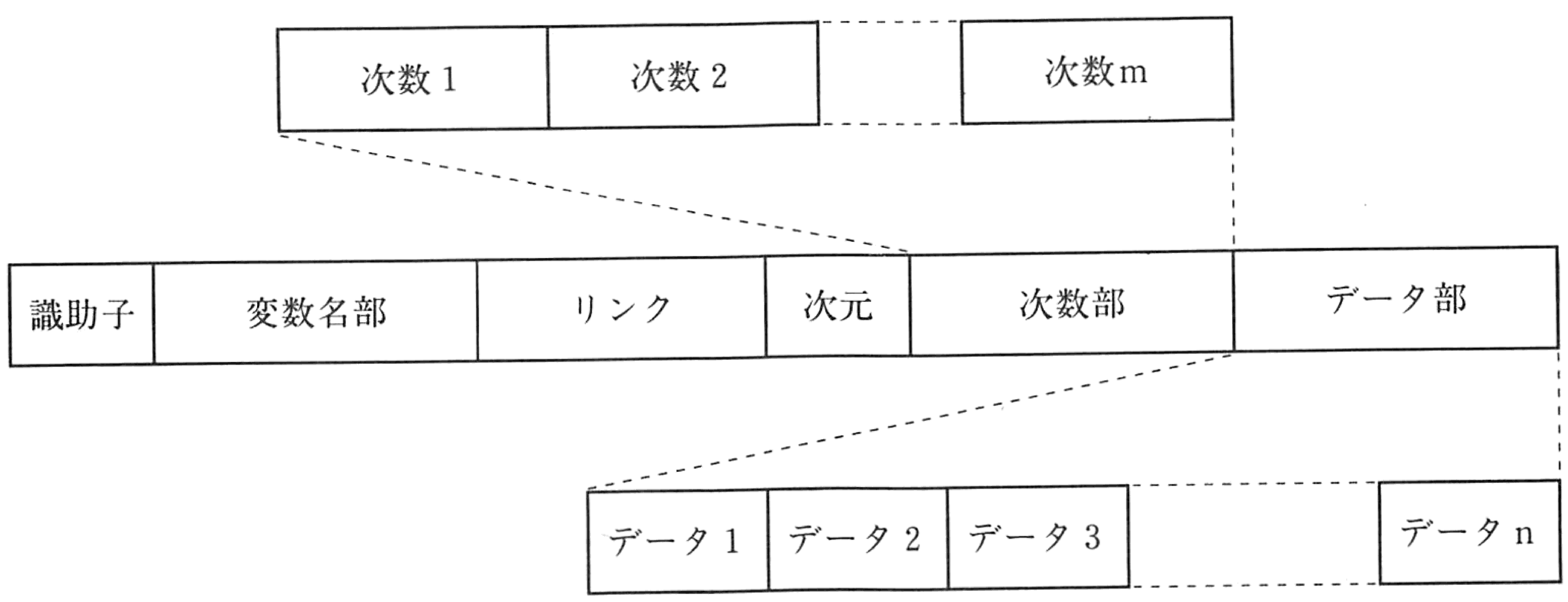


図4－1－3　配列変数の構造

4－1－3　配列変数の格納形式

配列変数の格納形式を図４－１－３に示します。このうち、識別子と変数名部は単純変数の場合と同じです。リンクというのはこれを含む以降の変数の長さです。これも変数サーチにおいて目的以外の変数をスキップするのに役立ちます。次元というのは添字の個数です。次数部には各次元の添字の最大値＋１が入ります(添字は０から始まることに注意)。最後のデータ部には各要素のデータが入ります。ここで注意しなければならないことは、次数部とデータ部のそれぞれの中味の格納順序がテキスト中に書かれた順番とは逆であるということです（図４－１－４)。変数サーチで配列を処理する際、出現順に添字をスタックへ積んでいき、添字が出尽くした後、今度は逆にスタックから一つずつ取り出して添字の処理をするため、このようなデータの並び方となります。

4－1－4　定数の格納方法

文字列定数はプログラムのリストと同じようにダブルクォーテーションマーク『"』に挟まれたまま格納されます。数値定数の場合は数値定数用中間言語の後に置かれます（図４－１－５)。ただし、０から９までの整数には、それぞれの数値を表すための専用の中間言語が定められています。また、１ＣＨ・１ＤＨ・１ＦＨに続く数値の具体的な格納方法はＦＡＣと同じですので、次節を参照して下さい。

4－1－5　ＦＡＣの構造

ＰＣ－８８０１で使用しているＣＰＵ：μＰＤ７８０Ｃ－１（Ｚ８０Ａコンパチブル）には浮動小数点演算の機能がありませんので、ＢＡＳＩＣインタプリタでは、この機能をソフトウェアで実現しています。この処理を行う際に中心的な役割を果すのが、ＦＡＣ（Floating point ACcumulator）と呼ばれる仮想レジスタです。ＦＡＣとその関連ワークエリアのメモリ上の配置を図４－１－６に示します。ただし、これらの使用方法は変数の型によって異なります（図４－１－７)。なお、ＦＡＣ２は倍精度実数型の２項演算を行うときにＦＡＣとペアで使われます。

（１）整数型の場合

整数は２バイトで表現されます。これは２の補数として扱われるので、とりうる値の範囲は－32768 ～ ＋32767 までとなります。ｎ進法におけるｎの補数というのは、各桁の数字をそれぞれｎ－１から引いたもので置き換え、最後に１を加えたものです。２進法の場合は、各ビットの値を反転してから１を足せばよいことになります（図４－１－８)。

○　プログラム

```
10  FOR  I=0  TO  2
20      FOR  J=0  TO  1
30          ABC%(I, J)=256*I+J
40      NEXT
50  NEXT
60  END
```

○　このプログラムを実行すると

識別子　| 02 |

変数名部　| 41 | 42 | 01 | C3 |
'A'　'B'　　'C'+80H

リンク　| 11 | 00 |
=17

次元　| 02 |

次数部　| 02 | 00 | 03 | 00 |

ABC%(2,1)

データ部　| 00 | 00 | 00 | 01 | 00 | 02 | 01 | 00 | 01 | 01 | 01 | 02 |
(0,0)　(1,0)　(2,0)　(0,1)　(1,1)　(2,1)

※　2バイトの整数はメモリ上で上位バイトと下位バイトが逆転することに注意!!

図4－1－4　配列変数の例

中　間　言　語（××は後続コード）		内　容
0 B	××　××	8進数が続く
0 C	××　××	16進数が続く
0 F	××	1バイト整数が続く
1 1		＝0
1 2		＝1
1 3		＝2
1 4		＝3
1 5		＝4
1 6		＝5
1 7		＝6
1 8		＝7
1 9		＝8
1 A		＝9
1 C	××　××	2バイト整数が続く
1 D	××　××　××　××	単精度実数が続く
1 F	××　××　××　××　××　××　××　××	倍精度実数が続く

図4－1－5　数値を表す中間言語

（2）単精度実数型の場合

単精度型の実数は4バイトで表現されます。このうち、仮数部に3バイト、指数部に1バイトが割り当てられます。

0以外の数値はすべて次の形で表すことができます（正規化表現）。

$\alpha = a \times 2^n, \quad 2^{-1} \leq |a| < 1$　　　ただしnは整数

このようにすると限られたビット数で最大の精度を保持することができます。上記のaを正規化表現の仮数部、nを同指数部と呼びます。ところで、仮数部aの最上位ビット（すなわち2^{-1}の位）は必ず“1”となりますのでこれは省略することができます。そこで、このビットには符号を割り当てています（“0”のとき正、“1”のとき負）。また、指数部nは実際の値に80Hだけ加えた値となっています。ただし、特に指数部が00Hの場合には、仮数部の値にかかわらず数値ゼロを表すものと約束します。このため、nの範囲は－127 ≦n≦127 となります。

図４－１－６には保護バイトというのがありますが、これは演算の精度を上げるためのＦＡＣの延長部分です。つまり、単精度実数型の場合には、実際の演算では32ビットの計算を行うわけです。また、演算ルーチンでは、符号が埋め込まれていると計算が複雑になるので、符号ビットにあるべき符号を符号バイトという領域に保存しておき、符号ビットのところには上述の最上位ビットの“１”を書き入れて計算します。

（３）倍精度実数型の場合

仮数部が７バイトとなること以外は単精度実数型と同じです。

ＥＡＢＤ	ＶＡＬＴＹＰ	ＦＡＣの型
ＥＣ３Ｃ	ＦＡＣ	ＦＡＣ保護バイト
ＥＣ３Ｄ		ＦＡＣ
ＥＣ４５		ＦＡＣ符号バイト
ＥＣ４６	ＦＬＧＯＶＣ	オーバーフローフラグ
ＥＣ４９	ＡＲＧ	ＦＡＣ２保護バイト
ＥＣ４Ａ		ＦＡＣ２

図４－１－６　ＦＡＣの構成

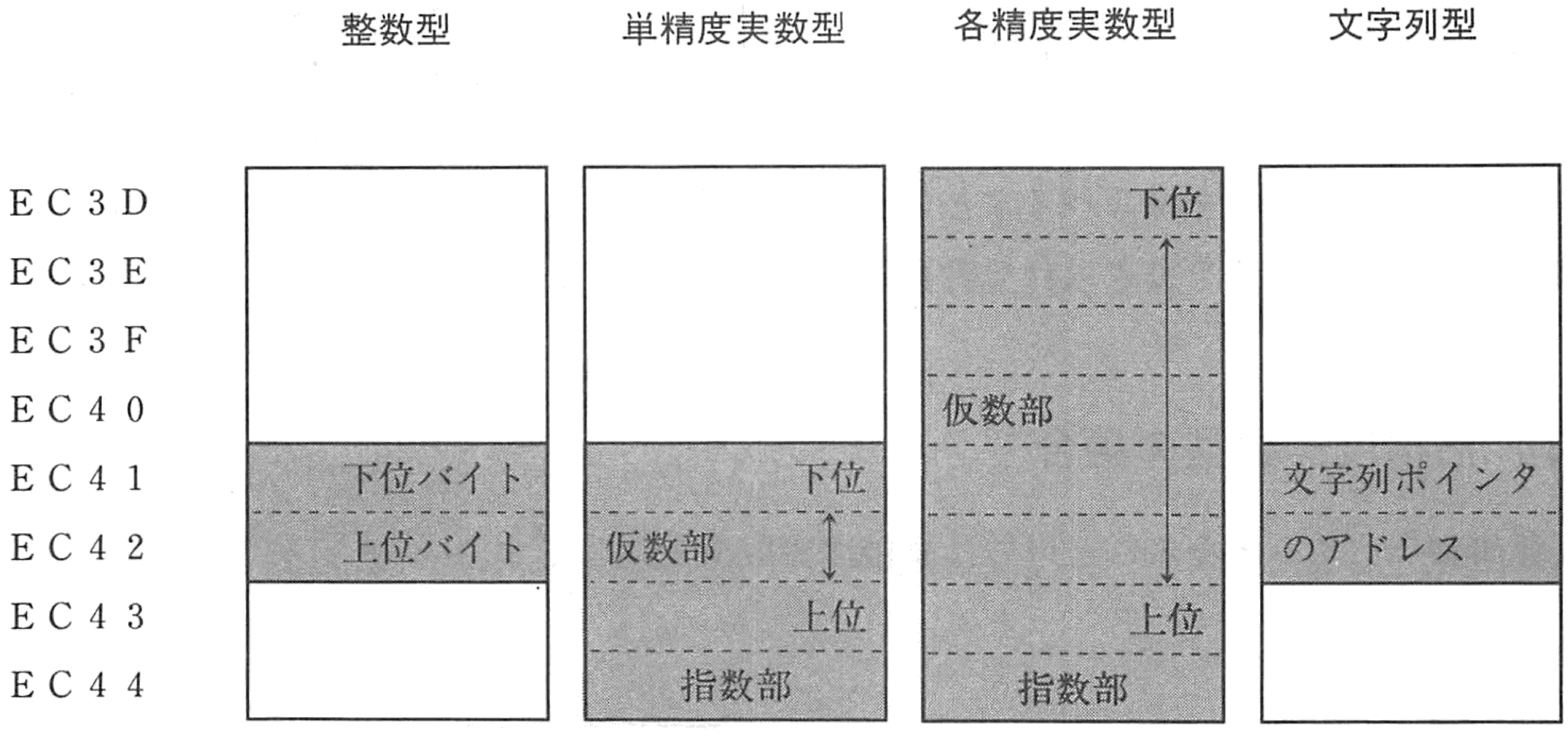


図４－１－７　型によるＦＡＣの使い方の違い

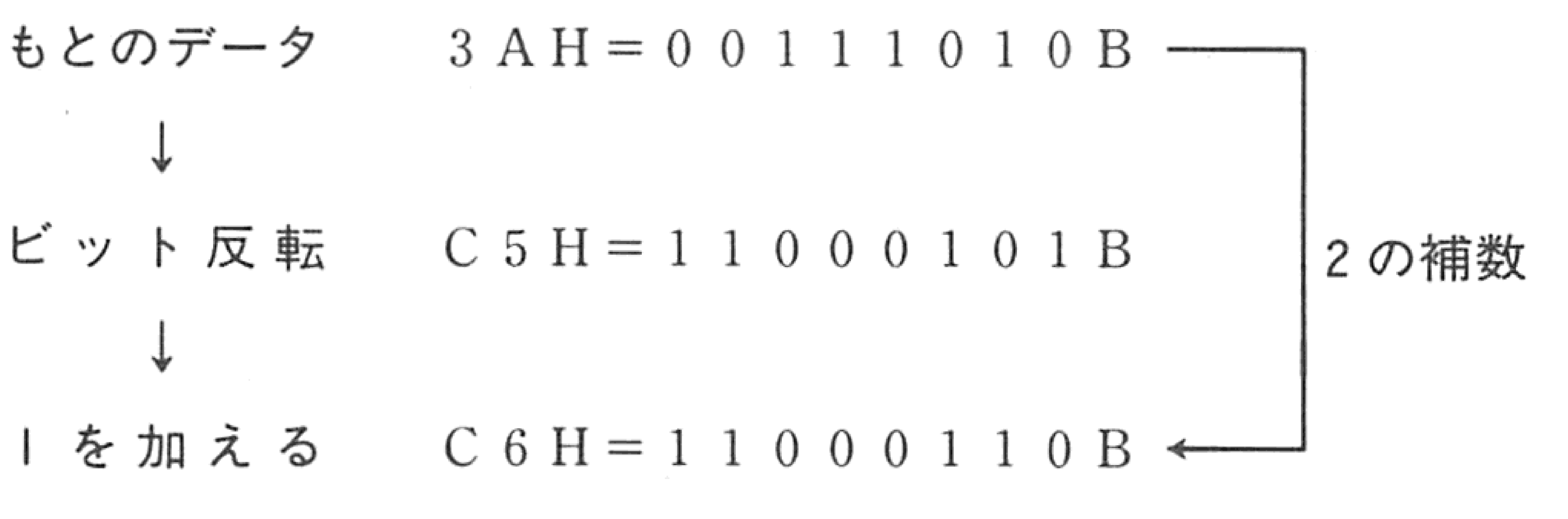


図４－１－８　２の補数

（４）文字列型の場合

文字列を扱う際にもＦＡＣは使われますが、実際に文字列が格納されるのは図４－１－１の文字列エリアです。この文字列の情報は、ストリングディスクプリタに格納されていますが、ＦＡＣにはこのストリングディスクプリタを指すアドレスが入ります。

ストリングディスクプリタは図４－１－９のような構成になっています。文字列長部には文字列の長さが収められ、アドレス部にはその文字列が実際に格納されている領域の先頭番地が入ります。

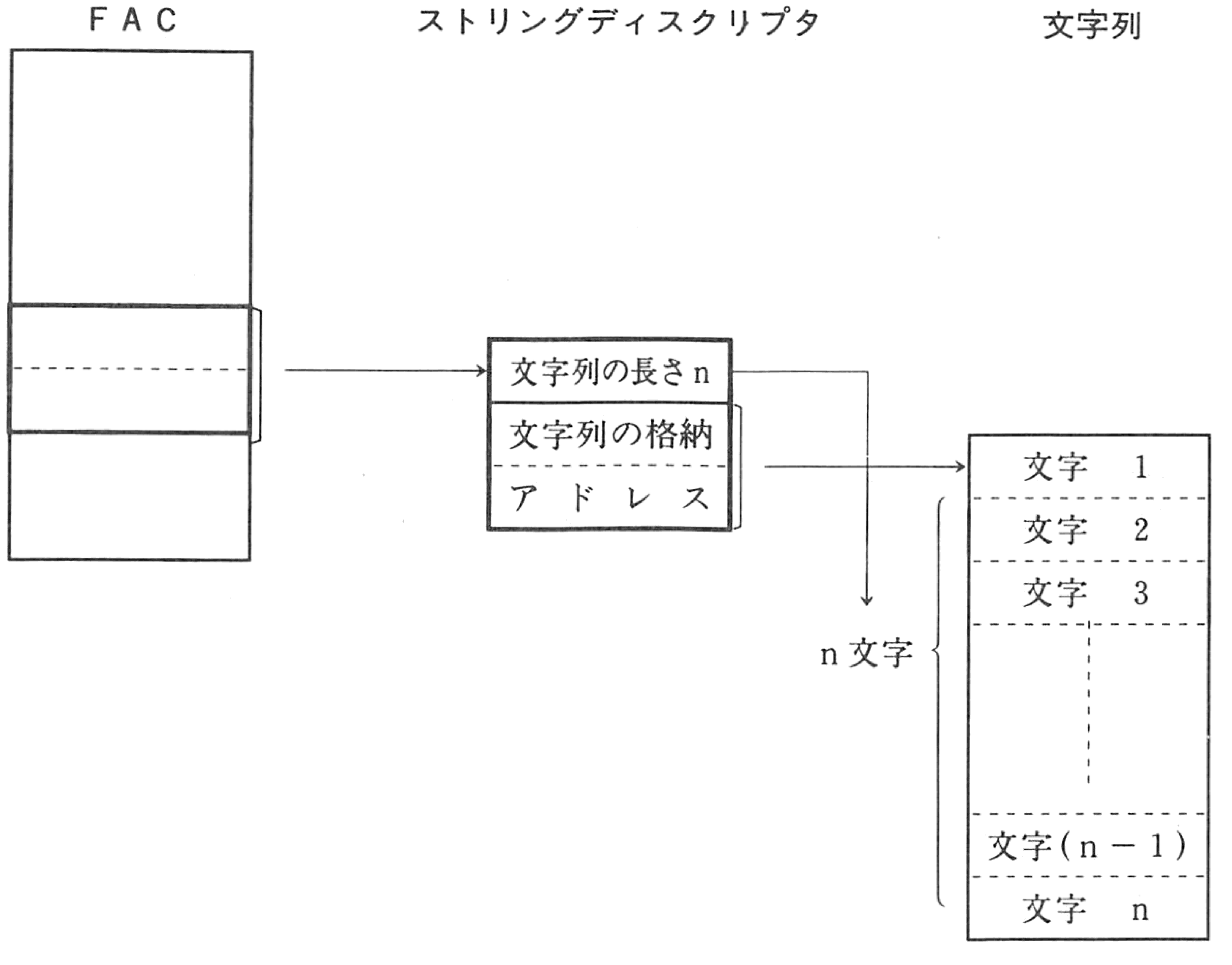


図4－1－9　ストリングディスクリプタ

4－2　LET文の処理

4－2－1　LET文ルーチン0C9CH：LET

領域	0C9CH～0D00H
機能	LET文の処理
破壊	A，F，B，C，D，E，H，L
入力	HLレジスタ：テキストポインタ
出力	DEレジスタ：左辺の変数のアドレス HLレジスタ：テキストの次の位置 左辺の変数　：右辺の演算結果
ワーク	E87AH（KBUF）　：中間言語バッファ（318バイト） EABDH（VALTYP）：FACの型 EAD0H（TEMPST）：ストリングディスクプリタスタック(3×10バイト) EAFDH（TEMP）　：左辺の変数アドレスの一時保管用 EB16H（LBLTAB）：ラベル領域開始アドレス EC3DH（FAC）　：FAC（8バイト）
ルーチン	0008H：1文字チェック 0020H：HLレジスタ－DEレジスタ 11D3H（FRMEVL）：式の評価 1796H：FACの型を変換する 20FCH：FACの値を変数に代入する 54D8H：演算により生成された文字列をストリングエリアにコピーする 56E7H：FACのストリングディスクプリタがストリングディスクプリタスタックにプッシュされているとき、そこからポップしてくる 5ACAH（PTRGET）：変数のアドレスを得る

まず、PTRGETを使ってテキストポインタから変数を一つ取りだします。これは『＝』の左辺になります。次に、『＝』があるのを確認します。これがないと、Syntax error となります。『＝』を検出したらFRMEVLを使って右辺にある式を評価しますが、右辺が左辺の型と異なるときは左辺の型に変換します。ここで、この型が文字列でなければ、FACに入っている右辺の計算結果を左辺の変数に代入してこのルーチンでの処理を終えます。文字列の場合には、右辺の文字列の格納場所をチェックします。これが中間言語バッファまたはファイルバッファの中にある場合、およびそのストリングディスクプリタがストリングディスクプリタス

タックの中にない場合は、目的のストリングをストリングエリアにコピーします。このときストリングエリアがいっぱいであれば、ガベージコレクションを行います。それから左辺の変数に右辺の演算結果（ＦＡＣ内）を代入します（図４－２－１）。

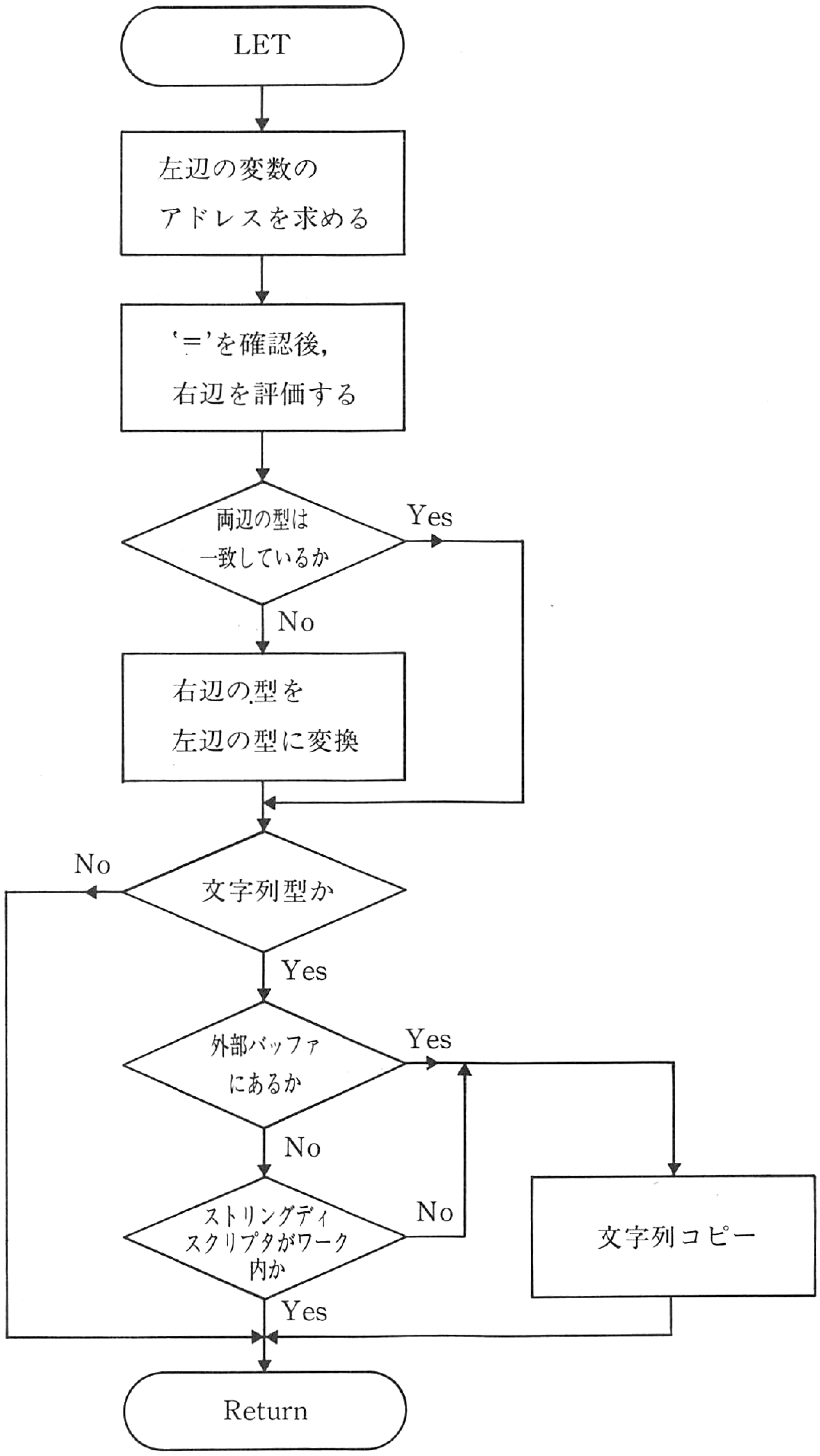


図４－２－１　ＬＥＴ文の処理

4−2−2　式の評価ルーチンＦＲＭＥＶＬ：１１Ｄ３Ｈ

領　域	１１Ｄ３Ｈ～１２Ａ９Ｈ
機　能	式の評価をする
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＨＬレジスタ：テキストポインタ
出　力	ＨＬレジスタ：更新されたテキストポインタ
ワーク	ＥＡＢＤＨ（ＶＡＬＴＹＰ）：ＦＡＣの型 ＥＡＦ３Ｈ（ＴＥＭＰ３）：テキストポインター時退避用 ＥＢ１０Ｈ（ＴＥＭＰ２）：テキストポインター時退避用 ＥＣ３ＤＨ（ＦＡＣ）：ＦＡＣ（８バイト） ＥＣ４６Ｈ（ＦＬＧＯＶＣ）：オーバーフローエラーフラグ
ルーチン	００８ＡＨ：演算子優先順位テーブル（12バイト） ０３９３Ｈ：Syntax error エントリ ０３Ｂ１Ｈ：Type mismatch エラーエントリ ０３Ｂ３Ｈ：String too long エラーエントリ １２６０Ｈ：比較演算子処理ルーチン １２７９Ｈ：単精度実数型のべき乗（＾）処理 １２８６Ｈ：論理演算子（ＡＮＤ，ＯＲ，ＸＯＲ，ＥＱＶ，ＩＮＰ）の処理 １３５０Ｈ（ＥＶＡＬ）：因子の評価 ２０ＣＤＨ：単精度実数型ＦＡＣをスタックに積む ２１Ａ０Ｈ：ＦＡＣを整数型にする ２２１４Ｈ：ＦＡＣを単精度実数型にする ２２５６Ｈ：ＦＡＣが文字列型でないときType mismatch エラー ４ＥＣ１Ｈ：スタックチェック ５４ＥＥＨ：ストリングエリアにＡレジスタ分の領域を設け、ＤＳＣＴＭＰにＡ・ＤＥレジスタに入っているストリングディスクプリタを退避する ５５２ＣＨ：ＤＳＣＴＭＰに入っているストリングディスクプリタを、ストリングディスクプリタスタックの最後のディスクプリタが指す領域に移し、ＦＡＣにこのディスクプリタの先頭アドレスを与える；その後、ストリングディスクプリタを更新する ５６８０Ｈ：文字列の結合（＋）処理ルーチン ５６Ｂ８Ｈ：ストリングディスクプリタをポップし、それを（ＤＥ）からの領域にコピーする

５６ＣＦＨ：ストリングディスクプリタスタックからディスクプリタをポップ
５６Ｄ０Ｈ：ストリングディスクプリタスタックからディスクプリタをポップ
ＥＤ３ＡＨ：フック（ＲＥＴ）
ＥＤ７５Ｈ：フック（ＪＰ　８Ｆ４１Ｈ）；倍精度実数型のべき乗（＾）処理

N_{88}－ＢＡＳＩＣでは、「式」は次のように定義されています。

＜因子＞［＜二項演算子＞＜式＞］

ここで、因子というのは式の最小単位で、定数（『＋』、『－』、『ＮＯＴ』が付いてもよい）、変数、関数、カッコで囲まれた式のことを指します。式を定義するのに自分自身を使っていることからもわかるように、このルーチンではリカーシブコール（再帰的呼び出し：あるルーチンが自分自身を呼び出すこと）を使っています。この処理と二項演算子の優先順位の扱いが、このルーチンのポイントとなります。以下、図４－２－２をもとにしてルーチンの説明をしていきます。

最初に、演算優先順位として００Ｈを設定し、スタックに入れます。各演算子には演算優先順位というのが決まっています(図４－２－３)。これは値が大きいほど、その演算子の優先度が高いことを示します。次に、Out of memory とならないことを確認した上で、因子の評価を行います（後述）。それから、その因子の次の文字（演算子）を調べます。これが比較演算子（『＜』、『＝』、『＞』）であったり、文字列の場合の『＋』である場合には、それぞれの処理へ分岐します。また、二項演算子でない場合は、このルーチンからリターンします。

以上のチェックのあと、すでにスタックに積んである演算優先順位と、新しい優先順位とを比較します。スタックにあったものの方が大きいか等しい場合にはここでリターンします。ここで、スタックに積んであった演算優先順位が００Ｈでない場合（リカーシブコールによってこのルーチンが現在実行されている場合）には、スタックに積んである戻り番地は各演算の処理アドレスです。００Ｈの場合には（この式評価ルーチンを呼ぶと最初に００Ｈが演算優先順位として設定されることに注意）式評価が完全に終わったので、このルーチンから本当にリターンします。新しいものの演算優先順位の方が大きい場合には、ＦＡＣ、ＦＡＣの型と演算優先順位、それぞれの演算を行うルーチンのアドレスをプッシュし、再びこの式評価ルーチンを実行します（リカーシブコール)。

５６８０Ｈからの文字列連結ルーチンは図４－２－４のようになっています。

優先順位をプッシュした後、ＦＡＣに入っているストリングディスクプリタポインタをプッシュします。それから、テキストの次の位置にある因子を評価します（これは文字列型でなければなりません)。次に、合成したあとの文字列の長さを計算し、それだけのスペースをストリングエリア内に確保します。そのあとで、連結する各ストリングのストリングディスクプリタをポップし、それらが指す文字列をストリングエリアへ順番にコピーしていきます（エリア上

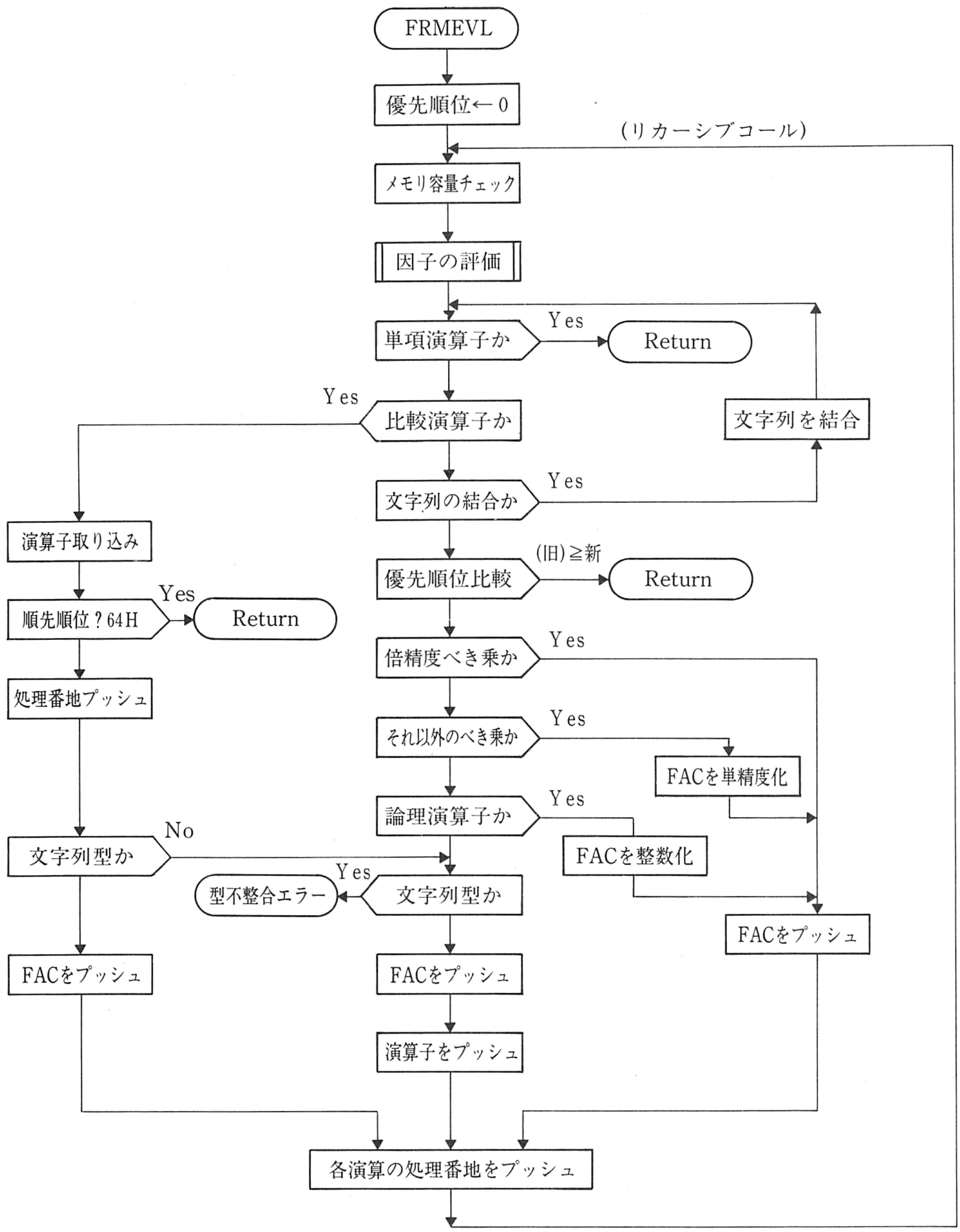

FRMEVL
優先順位← 0
(リカーシブコール)
メモリ容量チェック
因子の評価
単項演算子か
Yes
Return
Yes
比較演算子か
文字列を結合
文字列の結合か
Yes
演算子取り込み
(旧)≧新
優先順位比較
Return
Yes
順先順位？64H
Return
倍精度べき乗か
Yes
処理番地プッシュ
それ以外のべき乗か
Yes
FACを単精度化
論理演算子か
Yes
No
文字列型か
FACを整数化
Yes
型不整合エラー
文字列型か
FACをプッシュ
FACをプッシュ
FACをプッシュ
演算子をプッシュ
各演算の処理番地をプッシュ

図4−2−2　式の評価

に続けてコピーするのでこれで連結されたことになるのです)。最後に、この新しい文字列を指すストリングディスクプリタのアドレスをＦＡＣに入れ、このルーチンを終えます。

演算子	中間言語	優先順位
^(べキ乗)	Ｆ７	７Ｆ
－(負号)	Ｆ４	７Ｄ
＊	Ｆ５	７Ｃ
／	Ｆ６	７Ｃ
￥	ＦＥ	７Ｂ
ＭＯＤ	ＦＤ	７Ａ
＋	Ｆ３	７９
－(減算)	Ｆ４	７９
＜	Ｆ０	６４
＝	Ｆ１	６４
＞	Ｆ２	６４
ＮＯＴ	Ｅ３	５Ａ
ＡＮＤ	Ｆ８	５０
ＯＲ	Ｆ９	４６
ＸＯＲ	ＦＡ	３Ｃ
ＥＱＶ	ＦＢ	３２
ＩＭＰ	ＦＣ	２８

※　優先順位の値が大きいものほど優先度が高い。

※　＜，＝，＞は組み合わせて使用できる。

図４－２－３　各演算子の優先順位

Ｃ＄＝Ａ＄＋Ｂ＄の場合

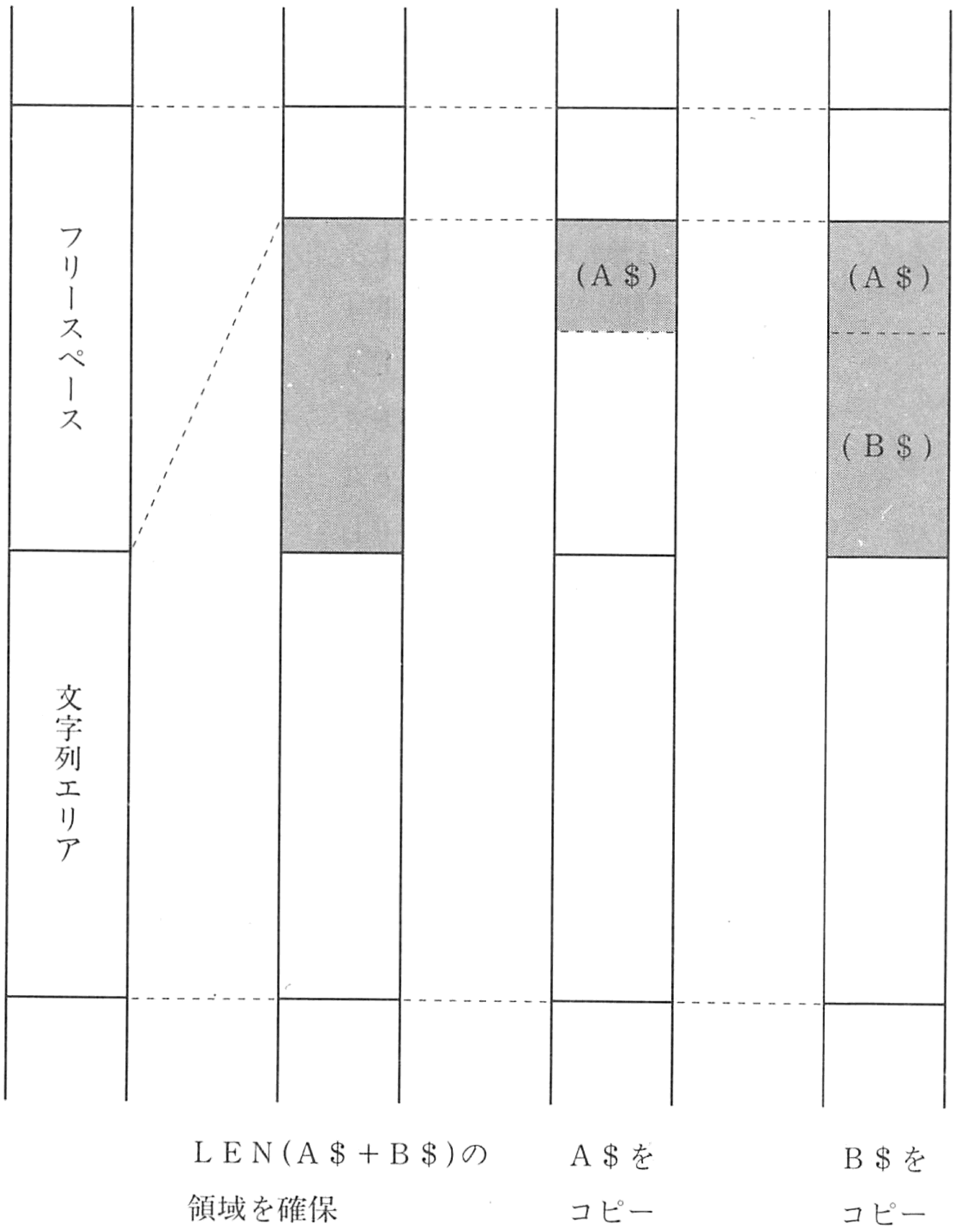


図４－２－４　文字列の連結

４－２－３　因子の評価ＥＶＡＬ：１３５０Ｈ

１３５ＥＨからのサブルーチンが中心となる処理系で、その前後ではＲＯＭの切り換えをしています。このルーチンのアルゴリズムを図４－２－５に示しました。関数の処理では、ジャンプテーブルを参照して各ルーチンへ分岐します。

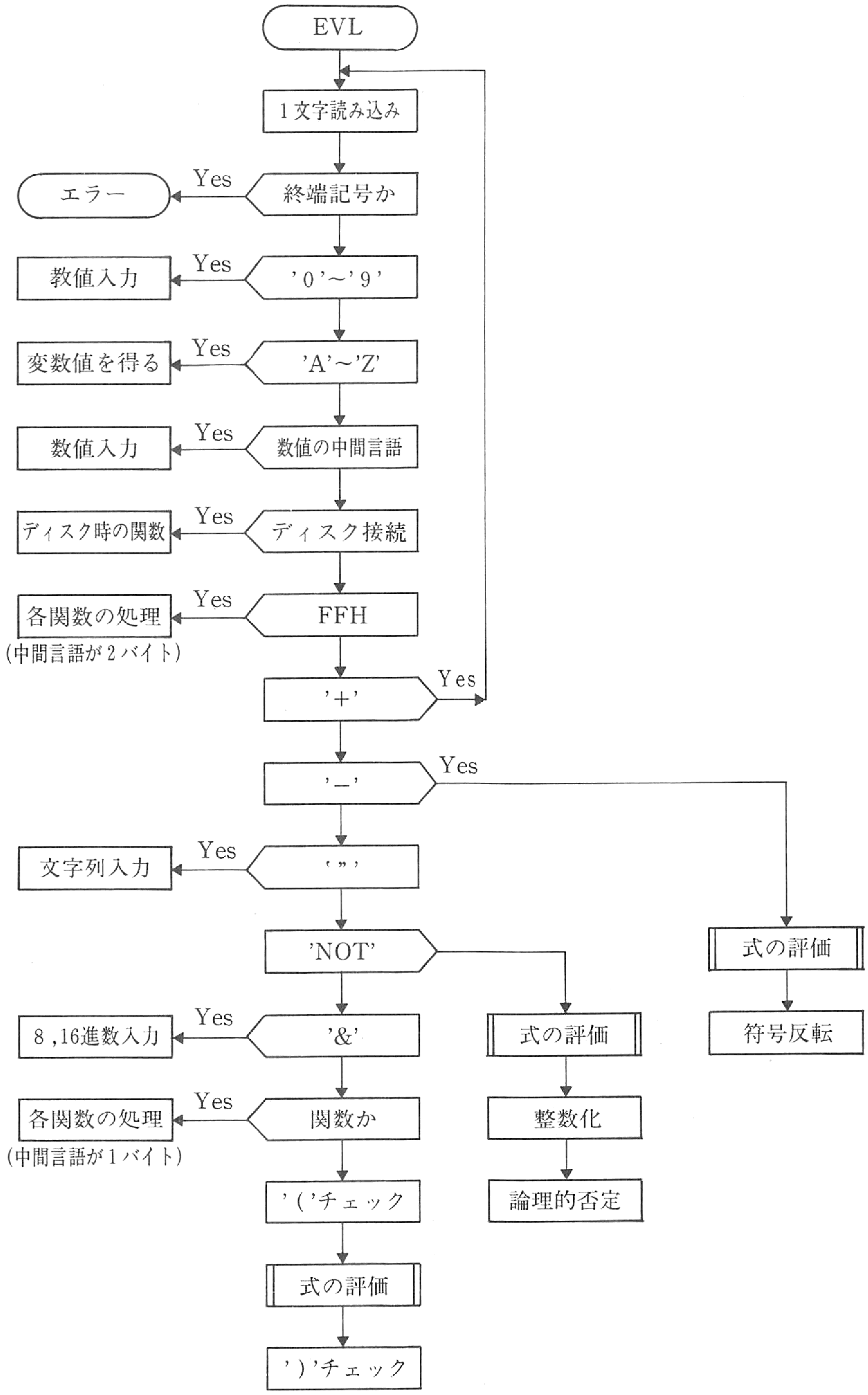

EVL
1文字読み込み
終端記号か
Yes
エラー
'0'~'9'
Yes
教値入力
'A'~'Z'
Yes
変数値を得る
数値の中間言語
Yes
数値入力
ディスク接続
Yes
ディスク時の関数
FFH
Yes
各関数の処理
(中間言語が2バイト)
'+'
Yes
'−'
Yes
'"'
Yes
文字列入力
式の評価
符号反転
'NOT'
式の評価
整数化
論理的否定
'&'
Yes
8,16進数入力
関数か
Yes
各関数の処理
(中間言語が1バイト)
'('チェック
式の評価
')'チェック

図4−2−5　因子の評価

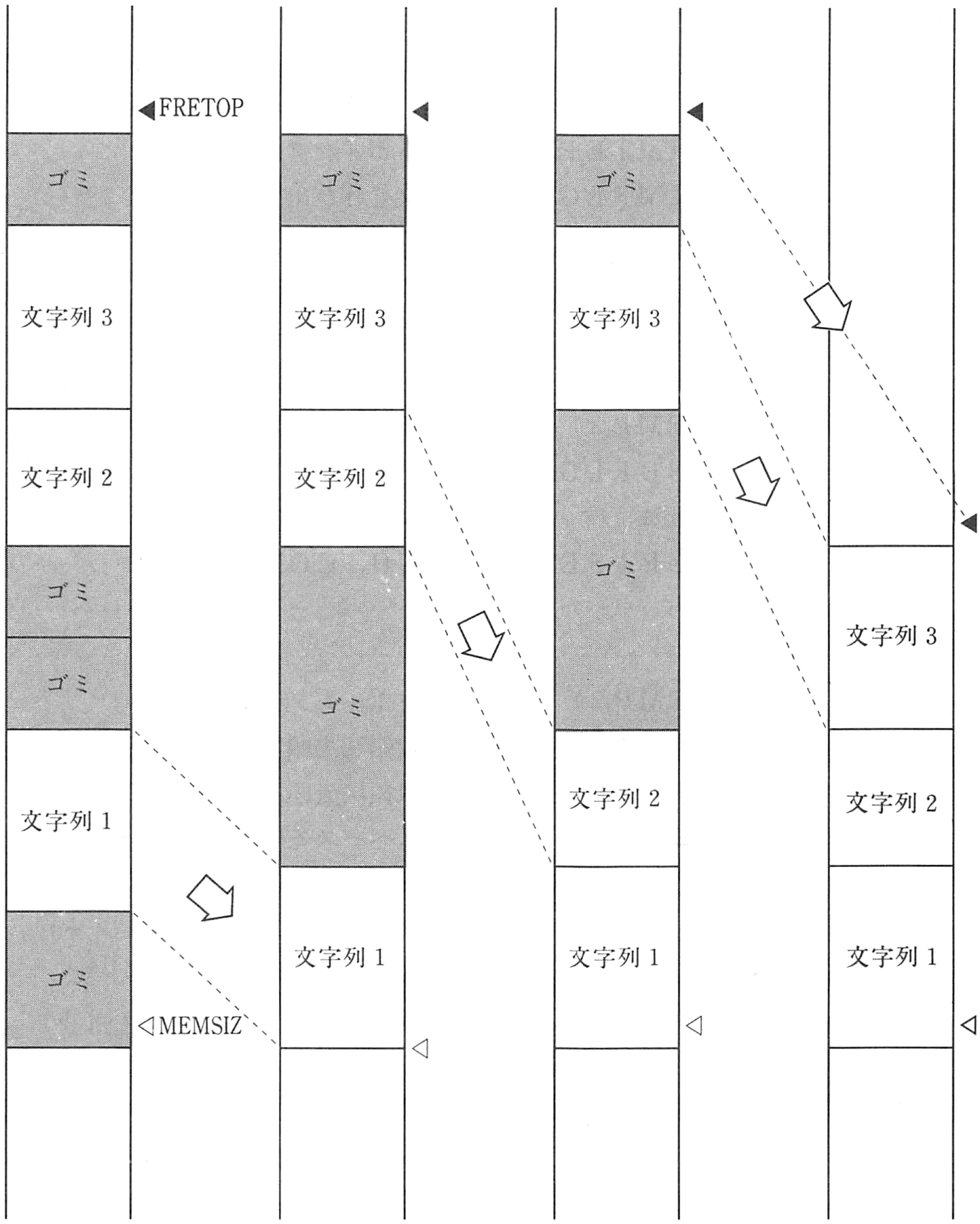
領　域	１３５０Ｈ～１４Ｆ５Ｈ
機　能	因子を評価する
破　壊	Ａ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＨＬレジスタ：テキストポインタ
出　力	ＨＬレジスタ：更新されたテキストポインタ
ワーク	Ｅ６４９Ｈ（ＥＲＲＦＬＧ）：エラーコード
	ＥＢ１０Ｈ（ＴＥＭＰ２）　：テキストアドレス一時保存用
	ＥＣ３ＤＨ（ＦＡＣ）　　：ＦＡＣの先頭アドレス
	ＥＣ４６Ｈ（ＦＬＧＯＶＣ）：オーバーフローエラーフラグ
ルーチン	０００８Ｈ：１文字比較
	００１０Ｈ：テキストから１文字入力
	０３９３Ｈ：Syntax error エントリ
	０３ＡＥＨ：Missing operand エラーエントリ
	０Ａ０ＤＨ：テキストから１文字読み込む
	０Ａ５ＦＨ：数値の入力
	０Ｂ０６Ｈ：Illegal function call エラーエントリ
	１１Ｄ１Ｈ：テキスト上の次の文字が『(』であることを確認したら、その次の式を評価する
	１１Ｄ３Ｈ：式の評価
	１５８９Ｈ：Ａレジスタの値を整数化し、ＦＡＣに入れる
	１３Ｆ２Ｈ：『(』のチェックの後、後続の式を評価し、さらに、そのあとに『)』があることを確認する
	１３Ｆ８Ｈ：『－』記号の処理
	１４０６Ｈ：変数から値を読み出し、ＦＡＣに入れる
	１４１５Ｈ：英小文字を英大文字に変換する
	１４２３Ｈ：『&』の処理（８進数、16進数の読み込み）
	１４ＡＤＨ：ＬＥＦＴ＄～ＭＫＤ＄(図２－２－２ ②参照）の処理
	１５１２Ｈ：ＮＯＴの処理
	１５８ＦＨ：ＵＳＲの処理
	１６００Ｈ：ＦＮの処理
	１８Ａ２Ｈ：テキストから１バイト定数を読み込み、ＦＡＣに入れる
	１８Ａ３Ｈ：１バイトの値を評価し、Ａレジスタに入れる
	２１１ＦＨ：各型ごとのＦＡＣ使用開始アドレスを求める
	２１ＦＤＨ：ＨＬレジスタの値を整数化し、ＦＡＣに入れる
	２２１４Ｈ：ＦＡＣを単精度実数型にする

２２５６Ｈ：ＦＡＣが文字列型であることを確認する
２４０２Ｈ：ＨＬレジスタの値を単精度実数型にする
２６ＢＣＨ：数値の入力
２Ｆ０ＢＨ：引数が省略された場合のＲＮＤ関数の処理
４６Ｆ８Ｈ（ＮＡＭＳＣＮ）：ファイルナンバーがＡレジスタで与えられる
　　　　　　　　　　　　　ファイルバッファのアドレスを求める
４ＢＡＣＨ：ＩＮＰＵＴの処理
５２１６Ｈ：Ａレジスタの値が英大文字であるかどうかを調べる
５４ＦＤＨ：ダブルクォーテーションマークの処理（文字列定数の読み込み）
５７Ｄ７Ｈ：ＩＮＳＴＲの処理
５ＡＡ３Ｈ：ＳＴＲＩＮＧ＄の処理
５ＡＣＡＨ：変数のアドレスを求める
５ＢＢ８Ｈ：変数のアドレスを求める
６ＡＢ０Ｈ：中間言語処理ジャンプテーブル（２）
６Ｂ０２Ｈ：中間言語処理ジャンプテーブル（３）
ＥＤ２７Ｈ：フック（ＪＰ　８ＦＡ４Ｈ）；ディスクＢＡＳＩＣのときの拡張処理（ＤＳＫＩ＄・ＩＮＰＵＴ・ＡＴＴＲ＄・ＵＳＲ）であれば、その処理を行う
ＥＤ２ＤＨ：フック（ＪＰ　８ＦＢＤＨ）；ディスクＢＡＳＩＣのときの拡張処理（ＥＯＦ・ＦＰＯＳ・ＬＯＣ・ＬＯＦ）であればその処理を行う
ＥＤ５ＡＨ：フック（ＪＰ　８ＦＢＤＨ）；倍精度関数のジャンプテーブルを求める

4−2−4　ガベージコレクション

領　域	５５９ＡＨ〜５６７ＦＨ
機　能	ガベージコレクションを行う
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	なし
出　力	なし
ワーク	Ｅ６９ＦＨ（ＴＲＭＣＭＤ）：ターミナルモードフラグ ００ＨのときＢＡＳＩＣモード；それ以外のときはターミナルモード ＥＡＣＣＨ（ＭＥＭＳＩＺ）：文字列エリアの開始アドレス ＥＡＣＥＨ（ＴＥＭＰＰＴ）：ストリングディスクプリタスタックポインタ（ＴＥＭＰＳＴのポインタ） ＥＡＤ０Ｈ（ＴＥＭＰＳＴ）：ストリングディスクプリタスタック（30バイト） ＥＡＦ１Ｈ（ＦＲＥＴＯＰ）：フリースペースの最大アドレス ＥＡＦ５Ｈ（ＴＥＭＰ８）：配列変数のリンク部保存用 ＥＢ１ＢＨ（ＶＡＲＴＡＢ）：単純変数領域の開始アドレス ＥＢ１ＤＨ（ＡＲＹＴＡＢ）：配列変数領域の開始アドレス ＥＢ１ＦＨ（ＳＴＲＥＮＤ）：フリースペースの開始アドレス ＥＢＡ５Ｈ（ＰＲＭＲＶ）：ＦＮ関数用引数テーブル２のポインタ ＥＣ０ＥＨ（ＡＲＹＴＡ２）：単純変数、ＦＮ関数引数のサーチ終了アドレス ＥＣ１１Ｈ（ＴＥＭＰ９）：ＦＮ関数用引数ポインタの退避用
ルーチン	００２０Ｈ：ＨＬレジスターＤＥレジスタ（レジスタ比較） ０Ｂ０６Ｈ：Illegal function call エラーエントリ ４ＥＢ６Ｈ：ＤＥレジスタからＢＣレジスタまでの文字列を（ＨＬ）に移す ５５Ｂ２Ｈ：ストリングディスクプリタスタックからディスクプリタを探す ５５ＣＣＨ：単純変数からストリングディスクプリタを探す ５５ＥＡＨ：ＦＮ関数の引数からストリングディスクプリタを探す ５６０ＢＨ：配列変数からストリングディスクプリタを探す ５６３ＣＨ：ＨＬレジスタが示すストリングディスクプリタのストリングがストリングエリアにある場合、スタック内のアドレスを更新する（本文参照） ５６５ＣＨ：探し出したストリングを後方に詰める ５Ｄ９ＤＨ：変数名の残りをスキップ

最初に、ＴＲＭＣＭＤを調べます。この値が００Ｈ（すなわちターミナルモード）場合には、Illegal function callエラーとなります。ＢＡＳＩＣモードであれば、ＦＲＥＴＯＰ中にＭＥＭＳＩＺの値を入れて実際の処理を始めます。まず、ストリングディスクプリタスタック、単純変数エリア、ＦＮ関数の引数、文字列配列からストリングディスクリプタをサーチしていき、その中から文字列の格納アドレスが最大のものを選び出します。それから、その文字列をＭＥＭＳＩＺ側にずらし、ストリングディスクリプタのアドレス部を直します。有効な文字列をすべて移動し終えるまで、以上の操作を続けます（図４－２－６）。



４－２－６　ガベージコレクション実行の様子

4－2－5　変数サーチ5ACAH：PTRGET

領　域	5ACAH～5DD4H
機　能	変数のアドレスを求める；未登録の変数であれば登録する
破　壊	A，F，B，C，D，E，H，L
入　力	HLレジスタ：テキストポインタ
出　力	HLレジスタ：更新されたテキストポインタ
ワーク	E850H（NAMCNT）：変数名－2（ただし、変数名が1文字のときは0とする）

E851H（NAMBUF）：変数名の3文字目以降の格納場所（38バイト）
E877H（NAMTEP）：テキストポインタ一時退避用
EABCH（DIMFLG）：DIM文で呼び出すときAFH；それ以外のときは00H
EABDH（VALTYP）：FACの型
※　なお、EABCHとEABDHとを組み合わせてリターンアドレスの一時退避用にも使われる
EAF3H（TEMP3）：登録する配列変数の位置を保存しておく
EAFBH（SUBFLG）：変数サーチをするときの変数の種類

普通の単純変数……………………………………………………00H
配列変数（ERASE，SEARCH，COMMON）…………01H
FN関数…………………………………………………………80H
FOR文…………………………………………………………64H

EB10H（TEMP2）：テキストポインタ一時退避用
EB1BH（VARTAB）：配列変数領域先頭アドレス
EB1DH（ARYTAB）：単純変数領域先頭アドレス
EB1FH（STREND）：フリースペース先頭アドレス
EB23H（DEFTBL）：型宣言文字が省略されたときに自動的に設定される変数の型（A～Zまでの26バイト）
EB3FH（PRMLEN）：FN関数引数テーブル1のバイト数
EB41H（PARM1）：FN関数引数テーブル1（100バイト）
EC1FH（OPTVAL）：OPTION　BASE文で指定した値
EC0DH（PRMFLG）：0以外のときFN関数引数サーチ中を表す
EC0EH（ARYTA2）：単純変数・FN関数の引数サーチ終了アドレス
EC10H（NOFUN）：00HのときFN関数の引数サーチを行わない
EC3DH（FAC）：FAC先頭アドレス

ルーチン　０010Ｈ：テキストから１文字読み込む
０020Ｈ：ＨＬレジスタ－ＤＥレジスタ
０030Ｈ：ＦＡＣの型を調べる。文字列型のときＺフラグが"１"となる
０393Ｈ：Syntax error エントリ
03Ａ2Ｈ：Duplicate Definition エラーエントリ
03Ｂ3Ｈ：エラーエントリ；Ｅレジスタにエラー番号を入れて呼ぶ
0Ｂ01Ｈ：正の整式を評価し、ＤＥレジスタに入れる
1ＦＡ5Ｈ：ＨＬレジスタをポップしリターンする
2312Ｈ：配列のバイト数を計算し、ＤＥレジスタに（ＤＥ＝ＢＣ×ＤＥ）；オーバーフローのときは Out of range エラー
4ＥＢ3Ｈ：ＤＥレジスタからＢＣレジスタまでを、ＨＬレジスタまで転送
4ＥＣ1Ｈ：スタックの深さをチェックする
4ＥＤ6Ｈ：Out of memory エラーエントリ
4ＥＥ4Ｈ：ストリングエリアの空きを調べる；もしスペースがなければ、ガベージコレクションをしてから再びチェックする
5215Ｈ：ＨＬレジスタの指す文字が英大文字以外のときキャリーフラグが"１"となる
5216Ｈ：Ａレジスタが英大文字以外のときキャリーフラグが"１"となる
5Ｃ0ＢＨ：変数の３文字目以降を比較する
5ＣＥ8Ｈ：Subscript out of range エラーエントリ
5ＤＡ6Ｈ：３文字目以降の変数名をセットする
5ＤＢＡＨ：ＨＬレジスタからＡバイト分をＥ851ＨからのＡバイトと比較（変数名の３文字目以降を比較する）
ＥＤ93Ｈ：フック（ＲＥＴ）

プログラムの実行中、変数が現れるたびに呼び出されるのがこのルーチンです（図４－２－７）。参照する変数がすでに登録されているものであれば、その値が格納されているアドレスをＤＥレジスタに返す一方、初めて現れる変数であれば、それを変数エリアに新たに登録してからそのアドレスを返します。ただし、新変数でも登録しない場合があります。ＥＲＡＳＥ，ＳＥＡＲＣＨ，ＣＯＭＭＯＮ，ＶＡＲＰＴＲ（<変数>）の各文で使われた場合と式の評価内で出てきた場合です。ここで注意が必要なのは式の評価の場合です。

```
10 PRINT A
20 PRINT VARPTR(A)
```

このプログラムを実行すると"０"が出力された後、Illegal function call が発生します。10行のような使われ方をした場合、Ａの値として「０」を返しはしますが、このときには変数

として登録されません。したがって、２０行でエラーとなります。

このルーチンのアルゴリズムは次のようになっています。まず、ＤＩＭＦＬＧにAレジスタの値をセットしたのち、変数名をテキストから取り込みます。この際、初めの2バイトはＢＣレジスタに入りますが、それ以降のものはＮＡＭＢＵＦに納められます。次に、変数の型をセットします。変数名の後に型宣言文字が付いていればそれが型となりますが、もし付いていなければ、ＤＥＦＴＢＬを参照して決められます。ＤＥＦＴＢＬは型宣言文（ＤＥＦＩＮＴ等）によってセットされますが、初期値としては整数型の値（０２Ｈ）が入ります。そのあとで、ＳＵＢＦＬＧによる分岐があります。これが０１Ｈとなるのは、ＳＥＡＲＣＨ，ＥＲＡＳＥ，ＣＯＭＭＯＮの場合だけです。これらのときは配列変数エリアのサーチをしますが、添字の計算はしません。ＳＵＢＦＬＧを００Ｈとするのは変数サーチの一般的な使い方です。このときは後ろに『(』または『[』があれば配列変数エリアを、なければ単純変数エリアを検索します。なお、ＣＯＭＭＯＮ文では初めＳＵＢＦＬＧ＝０１Ｈとしてこのルーチンを呼び、配列変数エリアを検索しますが、次には、ＳＵＢＦＬＧ＝００Ｈとして再びこのルーチンを呼び、今度は単純変数エリアをサーチします。

単純変数の処理と配列変数の処理に別れたあとは各変数の格納アドレスを求めますが、先にも書いたように、その後の処理はこのルーチンの呼び出しもとによって異なります。これについては図４－２－７を参照してください。

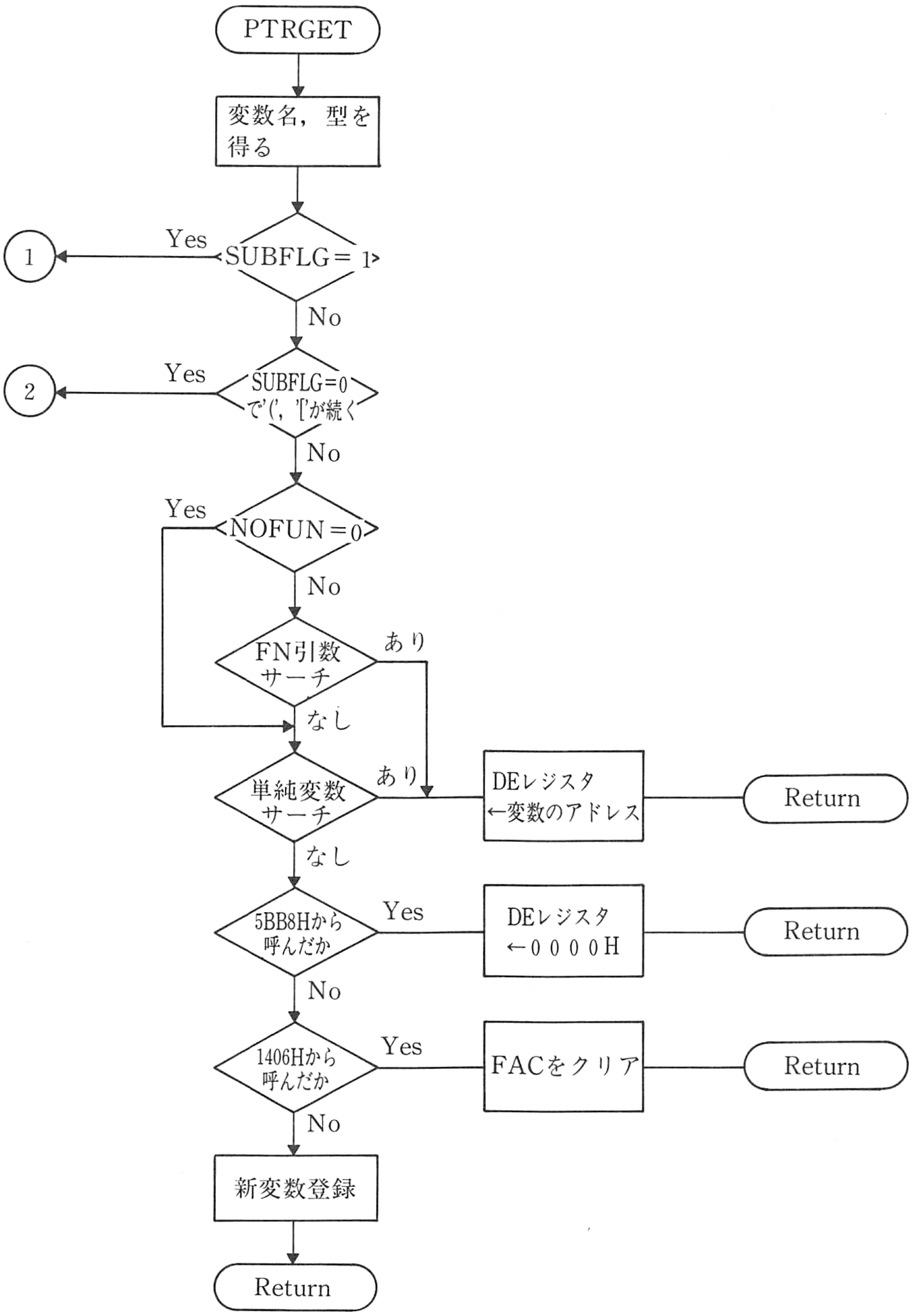


図4－2－7①　変数サーチ(2の1)

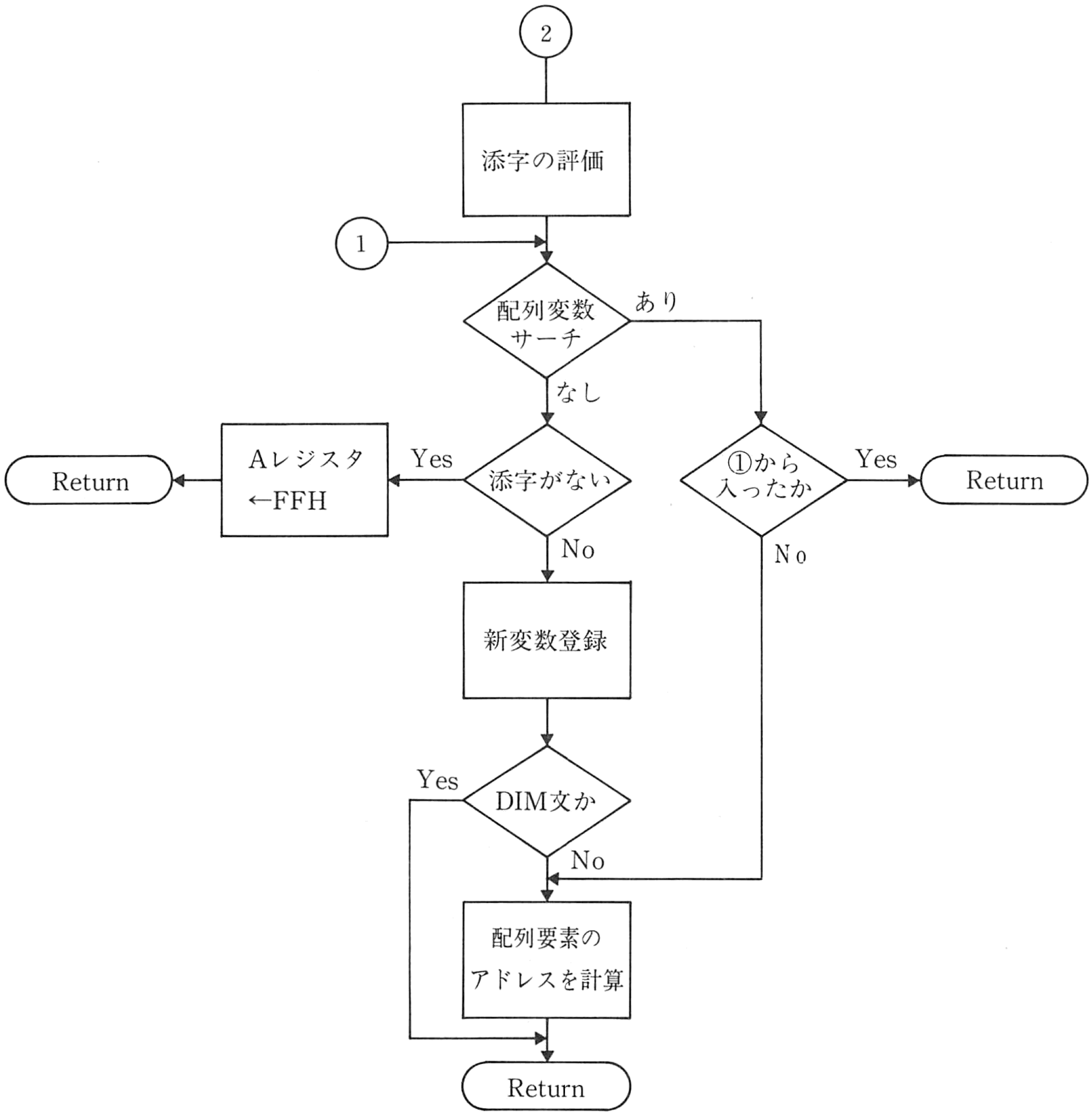


図4－2－7②　変数サーチ(2の2)

4－3　数値演算

（注意）　アドレスが［ＳＲ：××××Ｈ］という形で示されている場合、ＰＣ－８８０１mkIIＳＲでのアドレス（Ｅ２ＲＯＭ上）を表すものです。旧８８のＲＡＭ上に置かれていた処理系がＥ２ＲＯＭ上に移されて、メモリ上のロケーションが大幅に変更されたので注意が必要です。

4－3－1　数値演算の一般形式

数値演算の諸ルーチンを呼び出すときの一般的な形式を図４－３－１に示します。ここで整数型２項演算のときは注意が必要です。オーバーフローが起きなければＦＡＣとＨＬレジスタの両方に演算結果が格納されますが、オーバーフローを起こしたときは演算結果は単精度実数型に変換されて、ＦＡＣにのみ格納されます。

		第１引数	第２引数	演算結果
整　数　型	単項演算	ＨＬ		ＨＬ
	２項演算	ＤＥ	ＨＬ	ＦＡＣ，ＨＬ
単精度実数型	単項演算	ＦＡＣ		ＦＡＣ
	２項演算	ＢＣＤＥ	ＦＡＣ	ＦＡＣ
倍精度実数型	単項演算	ＦＡＣ		ＦＡＣ
	２項演算	ＦＡＣ	ＦＡＣ２	ＦＡＣ

図４－３－１　数値演算の一般形式

4－3－2　基本演算ルーチン

整数型比較ＩＣＯＭＰ：２１５ＦＨ

領　域　２１５ＦＨ～２１６ＢＨ

破　壊　Ａ

注　意　図４－３－２参照

単精度実数型比較ＦＣＯＭＰ：２１３４Ｈ

領　域	２１３４Ｈ～２１４ＢＨ
破　壊	A，F，H，L
注　意	図４－３－２参照

倍精度実数型比較ＤＣＯＭＰ：２１６ＥＨ

領　域	２１６ＥＨ～２１９８Ｈ
破　壊	A，F，B，C，D，H，L
注　意	図４－３－２参照

整数型加算ＩＡＤＤ：２３３ＡＨ

領　域	２３３ＡＨ～２３５７Ｈ
破　壊	A，F，B，C，D，E，H，L

整数型減算ＩＳＵＢ：２３２ＦＨ

領　域	２３２ＦＨ～２３３８Ｈ
破　壊	A，F，B，C，D，E，H，L

整数型乗算ＩＭＵＬＴ：２３５ＡＨ

領　域	２３５ＡＨ～２３ＡＡＨ
破　壊	A，F，B，C，D，H，L

整数型除算ＩＤＩＶ：２３ＡＢＨ

領　域	２３ＡＢＨ～２３Ｆ４Ｈ
破　壊	A，F，B，C，D，E，H，L
注　意	ＤＥレジスタには剰余が入る

整数型剰余ＩＭＯＤ：２４０ＣＨ

領　域	２４０ＣＨ～２４１ＢＨ
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
注　意	ＤＥレジスタには商が入る

整数型符号反転ＩＮＥＧ：２３Ｆ７Ｈ

領　域	２３Ｆ７Ｈ～２４０９Ｈ
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ

単精度実数型加算ＦＡＤＤ：１ＤＥＡＨ

領　域	１ＤＥＡＨ～１Ｅ９ＡＨ
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｈ，Ｌ

単精度実数型減算ＦＳＵＢ：１ＤＥ６Ｈ

領　域	１ＤＥ６Ｈ～１ＥＥ９Ｈ
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ

単精度実数型乗算ＦＭＵＬＴ：１Ｆ５３Ｈ

領　域	１Ｆ５３Ｈ～１ＦＡＢＨ
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ

単精度実数型除算ＦＤＩＶ：１ＦＢ７Ｈ

領　域	１ＦＢ７Ｈ～２０３６Ｈ
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ

実数型符号反転ＮＥＧ：２０ＡＢＨ

領　域	２０ＡＢＨ～２０Ｂ２Ｈ
破　壊	Ａ，Ｆ，Ｈ，Ｌ

注　意　単精度型でも倍精度型でも使える

倍精度実数型加算ＤＡＤＤ：２４２４Ｈ

領　域　２４２４Ｈ〜２４Ａ９Ｈ
破　壊　A，F，B，C，D，E，H，L

倍精度実数型減算ＤＳＵＢ：２４１ＤＨ

領　域　２４１ＤＨ〜２４２３Ｈ
破　壊　A，F，B，C，D，E，H，L

倍精度型実数乗算ＤＭＵＬＴ：２５５３Ｈ

領　域　２５５３Ｈ〜２５８ＣＨ
破　壊　A，F，B，C，D，E，H，L

倍精度実数型除算ＤＤＩＶ：２６２９Ｈ

領　域　２６２９Ｈ〜２６８９Ｈ
破　壊　A，F，B，C，D，E，H，L

	比　較	結果（Aレジスタの値）
整　数　型	ＤＥ＜ＨＬ	1
	ＤＥ＝ＨＬ	0
	ＤＥ＞ＨＬ	−1
単精度実数型	ＢＣＤＥ＜ＦＡＣ	1
	ＢＣＤＥ＝ＦＡＣ	0
	ＢＣＤＥ＝＞ＦＡＣ	−1
倍精度実数型	ＦＡＣ２＜ＦＡＣ	1
	ＦＡＣ２＝ＦＡＣ	0
	ＦＡＣ２＝ＦＡＣ	−1

図４−３−２　数値比較ルーチンの実行結果

4－3－3　型変換ルーチン

ＦＡＣ整数型化ＦＲＣＩＮＴ：２１Ａ０Ｈ

領　域	２１Ａ０Ｈ～２１ＢＢＨ
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｈ，Ｌ

ＦＡＣ単精度型実数化ＦＲＣＳＮＧ：２２１４Ｈ

領　域	２２１４Ｈ～２２１ＢＨ
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ

ＦＡＣ倍精度型実数化ＦＲＣＤＢＬ：２２３ＥＨ

領　域	２２３ＥＨ～２２４５Ｈ
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ

単精度実数型→整数型変換ＣＯＮＩＳ：２１ＢＣＨ

領　域	２１ＢＣＨ～２１ＦＣＨ
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ

整数型→単精度実数型変換ＣＯＮＳＩ：２２２ＦＨ

領　域	２２２ＦＨ～２２３ＤＨ
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ

単精度実数型→倍精度実数型変換ＣＯＮＤＳ：２２４６Ｈ

領　域	２２４６Ｈ～２２５５Ｈ
破　壊	Ａ，Ｆ，Ｈ，Ｌ

倍精度実数型→単精度実数型変換ＣＯＮＳＤ：２２１ＣＨ

領　域	２２１ＣＨ～２２２ＣＨ
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｈ，Ｌ

４－３－４　多項式の計算

N_{88}－ＢＡＳＩＣでは近似多項式から初等関数の値を求めていますが、この処理を行うのがこのルーチンです。計算する多項式としては、一般に次のようなものが考えられます。

【１】　$C_0+C_1x+C_2X^2+C_3x^3+\cdots\cdots+C_nx^n$

【２】　$C_0x+C_1x^3+C_2x^5+C_3x^7+\cdots\cdots+C_nx^{2n+1}$

【３】　$C_0+C_1x^2+C_2x^4+C_3x^6+\cdots\cdots+C_nx^{2n}$

【１】が一般的な形ですが、奇関数では【２】、偶関数では【３】となります。このうちインタプリタが用意しているのは【１】と【２】だけです。偶関数である cos x の展開式の計算など【３】が必要なときは、【１】式にx^2を代入すれば求められます（ただし、実際の cos x の計算では sin x から求める）。なお上の式のxにはＦＡＣの値が使われます。また係数C_0～C_nは図４－３－３のような形式でメモリ上に置かれます。

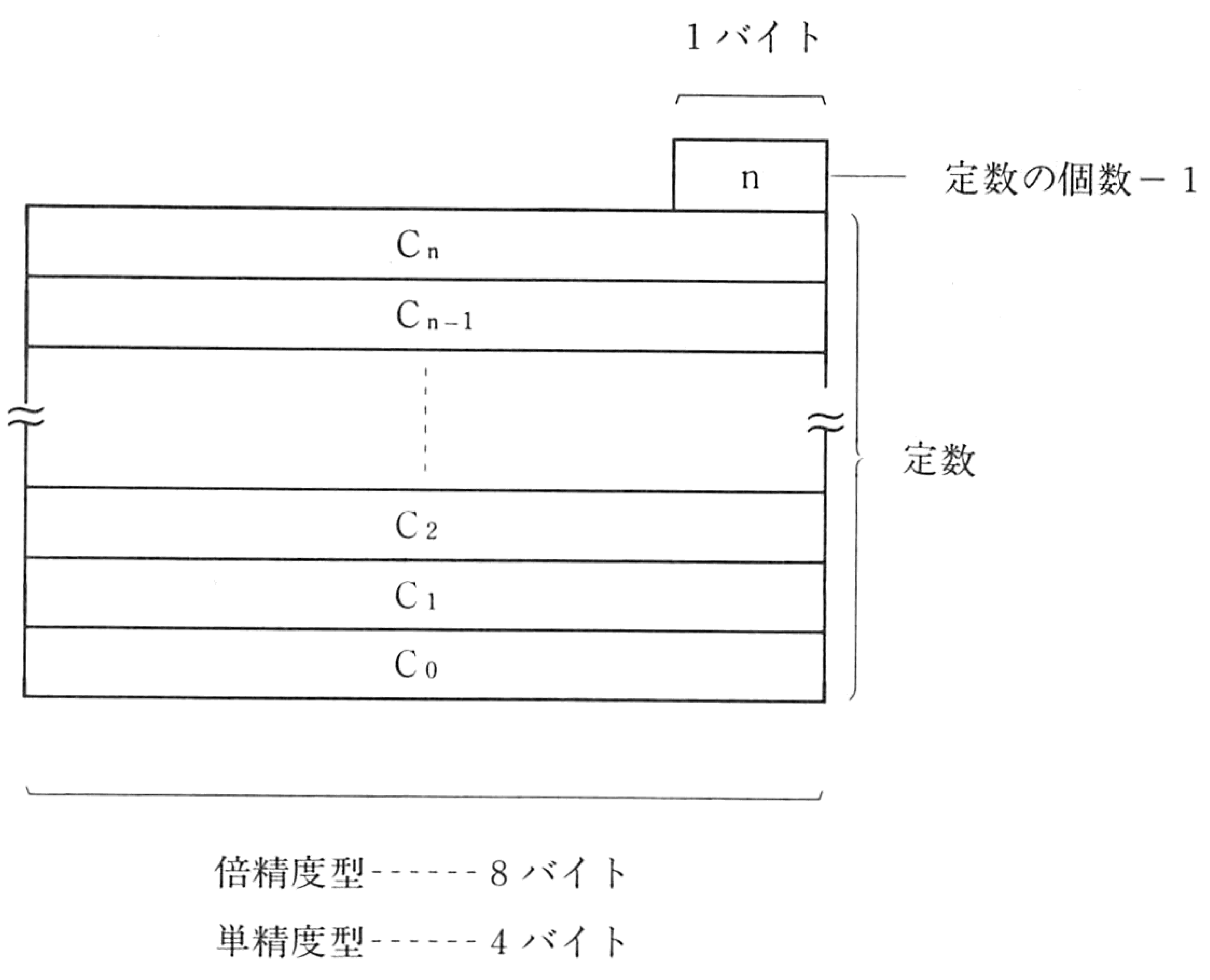


図４－３－３　多項式係数領域の構成

単精度実数型【1】POLY：2EE7H

領　域	2EE7H～2F06H
機　能	多項式【1】の計算
破　壊	A，F，B，C，D，E，H，L
入　力	HLレジスタ：多項式係数領域の先頭アドレス
出　力	FAC：演算結果
ワーク	なし
ルーチン	1DE9H（FADD）：単精度実数型の加算 1F53H（FMULT）：単精度実数型の乗算 20CDH：スタックにFACを積む 20DAH：HLレジスタの指すところから4バイトをFACに積む 20EBH：HLレジスタの指すところから4バイトをBCDEレジスタに入れる

次のように式を解釈し、カッコの内側から計算していきます。

$$C_0+C_1x+C_2x^2+\cdots\cdots+C_{n-1}x^{n-1}+C_nx^n$$

$$=C_0+x\ (C_1+x\ (C_2+\cdots\cdots+x\ (C_{n-1}+C_nx)\ \cdots\cdots))$$

これはつぎの漸化式をP_nまで求めることに帰着します（Hornor法)。

$$P_0=C_n$$

$$P_k=P_{k-1}+x\times C_{n-k}\quad (k=1,\ 2,\ \cdots\cdots\cdots,\ n)$$

単精度実数型【2】POLYX：2ED8H

領　域	2ED8～2EE6H
機　能	多項式【2】の計算
破　壊	A，F，B，C，D，E，H，L
入　力	HLレジスタ：多項式係数領域の先頭アドレス
出　力	FAC：演算結果
ワーク	なし
ルーチン	1F53H（FMULT）：単精度実数型の乗算 20CDH：FACをスタックに積む 20E8H：FACの値をBCDEレジスタにコピーする 2398H：スタックから4バイトをBC・DEレジスタにポップして、FMULTを実行する

2EE7H（POLY）：多項式【1】の計算

POLYルーチンで定義される多項式をP（x）とすると、xP（x^2）により計算されます。

$$C_0x+C_1x^3+C_2x^5+\cdots\cdots+C_nx^{2n+1}$$
$$=xC_0+xC_1(x^2)+xC_2(x^2)^2+\cdots\cdots+xC_n(x^2)^n$$

倍精度実数型【1】DPOLY：92B0H［SR：6B82H］

領　域	92B0H～92D3H［SR：6B82F～6BA5H］
機　能	倍精度実数型の多項式の計算【2】
破　壊	A，F，B，C，D，E，H，L
入　力	FAC：引数 HLレジスタ：係数領域の先頭アドレス
出　力	FAC演算結果
ワーク	EC52H（FBUFFR）：計算結果の一時退避用
ルーチン	9253H［SR：6B25H］：FACに（HL）の番地の値をかける 9273H［SR：6B45H］：FAC2←（HL） 9278H［SR：6B4AH］：HLレジスタの指す番地から倍精度実数を取り込み、FACに入れる 927DH［SR：6B4FH］：FACの値を（HL）の番地へ移す 9424H［SR：6CF6H］：FAC←FAC＋FAC2

※単精度実数型【1】と同じです。

倍精度実数型【2】DPOLYX：929BH［SR：6B6DH］

領　域	929BH～92AFH［SR：6B6DH～6B81H］
機　能	倍精度実数型の多項式【2】を計算する
破　壊	A，F，B，C，D，E，H，L
入　力	FAC：引数 HLレジスタ：多項式係数領域先頭アドレス
出　力	FAC：演算結果
ワーク	EC52H（FBUFFR）：計算結果の一時退避用

ルーチン	２５５３Ｈ（ＤＭＵＬＴ）　：倍精度実数型の乗算
	９２Ｂ０Ｈ［ＳＲ：６Ｂ８２Ｈ］（ＤＰＯＬＹ）：倍精度型多項式計算【２】
	９２Ｄ９Ｈ［ＳＲ：６ＢＡＢＨ］：ＦＡＣをスタックにプッシュする
	９２Ｅ９Ｈ［ＳＲ：６ＢＢＢＨ］：スタックからＦＡＣ２にポップしてくる
	９３０ＦＨ［ＳＲ：６ＢＥ１Ｈ］：ＦＡＣ←ＦＡＣ×ＦＡＣ

※単精度実数型【２】と同じです。

４－３－５　逆正接関数

単精度実数型ＡＴＮ：８９Ｂ３Ｈ［ＳＲ：６５５４Ｈ］

領　域	８９Ｂ３Ｈ～８９Ｄ６Ｈ［ＳＲ：６５５４Ｈ～６５７７Ｈ］
機　能	単精度実数型の逆正接関数を求める
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＦＡＣ：引数
出　力	ＦＡＣ：演算結果
ワーク	３００ＦＨ：π／２の値（４バイト）
	８９Ｄ７Ｈ［ＳＲ：６５７８Ｈ］：多項式の係数
	ＥＣ３ＤＨ（ＦＡＣ）：ＦＡＣの先頭アドレス
ルーチン	００２８Ｈ：ＦＡＣの符号を調べる
	１１Ｄ３Ｈ：ＨＬの指す定数４バイトからＦＡＣを引きいてＦＡＣに入れる
	１ＦＢ７Ｈ（ＳＤＩＶ）　：単精度実数型の除算
	２０ＡＢＨ（ＮＥＧ）　：ＦＡＣの符号を反転する
	２Ｅ００Ｈ：符号反転ルーチンのエントリをスタックに積む
	２ＥＤ８Ｈ（ＰＯＬＹＸ）：多項式計算ルーチン【２】
	ＥＥＣ８Ｈ：フック（ＪＰ　８９Ｂ３Ｈ）

以下の操作を施すことによって引数Ｘの値を０以上１未満としてから近似多項式に代入し、後でその補正をします。

（１）　Ｘ＜０の場合

ＡＴＮ（Ｘ）＝－ＡＴＮ（－Ｘ）という関係式を用いる。Ｘの値を正に変えてから近似多項式に代入し、その結果をさらに負にする。

（２）　｜Ｘ｜≧１の場合

ＡＴＮ（Ｘ）＝π／２－ＡＴＮ（１／Ｘ）という関係を利用する。Ｘの逆数を多項式に代入し、後で結果をπ／２から引く。

なお、Ｘの値によっては（１）と（２）の両方を満たす場合もあれば、どちらにも属しない場合もあるので、場合分けは全部で４通りとなります。

ここで計算する多項式は次のとおりです。

$$P(x)= C_0x^1+C_1x^3+C_2x^5+C_3x^7+C_4x^9+C_5x^{11}+C_6x^{12}+C_7x^{15}+C_8x^{17}$$

$C_0=\ \ 1.00000$

$C_1=-0.333332$

$C_2=\ \ 0.199936$

$C_3=-0.142089$

$C_4=\ \ 0.106563$

$C_5=-0.0752896$

$C_6=\ \ 0.0429096$

$C_7=-0.0161657$

$C_8=\ \ 0.00286623$

倍精度実数型ＤＡＴＮ：９４８ＢＨ［ＳＲ：６Ｄ５ＤＨ］

領　域	９４８ＢＨ～９４Ｄ７Ｈ［ＳＲ：６Ｄ５ＤＨ～６ＤＡ９Ｈ］
機　能	倍精度実数型の逆正接関数を求める
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＦＡＣ：引数
出　力	ＦＡＣ：演算結果
ワーク	９３２６Ｈ［ＳＲ：６ＢＦ８Ｈ］倍精度実数型逆正接関数の係数領域 ９３８９Ｈ［ＳＲ：６Ｃ５ＢＨ］：定数1.0 ９４７３Ｈ［ＳＲ：６Ｄ４５Ｈ］：定数$\sqrt{3}$ ９４８３Ｈ［ＳＲ：６Ｄ５５Ｈ］：定数ＴＡＮ（π／１２） ９４Ｄ８Ｈ［ＳＲ：６ＤＡＡＨ］：ＦＡＣ←（π／２－ＦＡＣ）を実行するルーチンのエントリアドレス ＥＣ３ＤＨ（ＦＡＣ）：ＦＡＣの先頭アドレス
ルーチン	２６２９Ｈ（ＦＤＩＶ）：単精度実数型の除算 ９２４７Ｈ［ＳＲ：６Ｂ１９Ｈ］：ＦＡＣ←ＦＡＣ＋（ＨＬ） ９２４ＤＨ［ＳＲ：６Ｂ１ＦＨ］：ＦＡＣ←ＦＡＣ－（ＨＬ） ９２５３Ｈ［ＳＲ：６Ｂ２５Ｈ］：ＦＡＣ←ＦＡＣ×（ＨＬ） ９２５９Ｈ［ＳＲ：６Ｂ２ＢＨ］：ＡＦ←ＳＧＮ（ＦＡＣ－（ＨＬ））

９２８９Ｈ［ＳＲ：６Ｂ５ＢＨ］：ＦＡＣ２←ＦＡＣ
９２９３Ｈ［ＳＲ：６Ｂ６５Ｈ］：ＮＥＧルーチン（ＦＡＣの符号反転）を実行した後、ＮＥＧのエントリアドレスをスタックにプッシュする
（このＤＡＴＮルーチンからリターンするときＮＥＧを実行するため）
９２９ＢＨ［ＳＲ：６Ｂ６ＤＨ］（ＤＰＯＬＹＸ）：倍精度型多項式【２】
９２Ｄ９Ｈ［ＳＲ：６ＢＡＢＨ］：ＦＡＣをスタックにプッシュする
９２Ｅ９Ｈ［ＳＲ：６ＢＢＢＨ］：スタックからＦＡＣ２にポップする
９２ＦＥＨ［ＳＲ：６ＢＤ０Ｈ］：ＡＦ←ＳＧＮ（ＦＡＣ）
９３１５Ｈ［ＳＲ：６ＢＥ７Ｈ］：スタックにプッシュされている値とＦＡＣの値を交換する
９４Ｅ１Ｈ［ＳＲ：６ＤＢ３Ｈ］：ＦＡＣ←（ＦＡＣ＋π／６）を実行するルーチンのエントリアドレス

まず、次のチェックをします。

（１）　Ｘ＜０のとき
ＡＴＮ（Ｘ）＝－ＡＴＮ（Ｘ）であるので、Ｘの符号を正にしてから以下にすすみ、それによって得られた結果の符号を再び最後で反転する。

（２）　Ｘ≧１のとき
ＡＴＮ（Ｘ）＝π／２－ＡＴＮ（１／Ｘ）を利用する。Ｘ＝１／Ｘとして以下に進みあとで補正する。

（３）　Ｘ＞ＴＡＮ（π／１２）＝0.2679491924311227のとき
$ATN(X)=\frac{\pi}{6}+ATN\left(\frac{X\times\sqrt{3}-1}{X+\sqrt{3}}\right)$を利用する。$X=\frac{X\times\sqrt{3}-1}{X+\sqrt{3}}$としてから近似多項式を計算し、あとでπ／６を加える。

以上により０≦Ｘ≦ＴＡＮ（π／１２）としたのち、次の近似多項式に代入します。

$$P(x)= C_0x+C_1x^3+C_2x^5+C_3x^7+C_4x^9+C_5x^{66}+C_6x^{13}+C_7x^{15}+C_8x^{17}$$

$C_0=$　1.000000000000000
$C_1=-$0.3333333333332993
$C_2=$　0.1999999999872945
$C_3=-$0.1428571410282555
$C_4=$　0.1111109789805105
$C_5=-$0.09090371141910740
$C_6=$　0.07679368690660000
$C_7=-$0.06483193510303000

C_8＝　0.04438951571800000

4−3−6　指数関数

単精度実数型ＥＸＰ：２Ｅ６ＥＨ

領　域	２Ｅ６ＥＨ～２ＥＡＦＨ
機　能	単精度実数型の指数関数を求める
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＦＡＣ　：引数
出　力	ＦＡＣ　：演算結果
ワーク	２ＥＢＢＨ：多項式係数領域先頭アドレス ＥＣ３ＤＨ（ＦＡＣ）　：ＦＡＣの先頭アドレス
ルーチン	１ＤＥ６Ｈ（ＦＳＵＢ）　：単精度実数型の減算 １Ｆ５３Ｈ（ＦＭＵＬＴ）：単精度実数型の乗算 ２０ＣＤＨ：ＦＡＣをスタックに積む ２２９ＦＨ：ＦＡＣを整数化する ２８３３Ｈ：オーバーフローの処理 ２Ｅ６ＥＨ：$\log_2 e$の値をＢＣＤＥレジスタに代入（ここから６バイト）

コンピュータは内部で２進法による演算を行いますので、２の整数乗の乗算は指数部の足し算だけで済ませることができ、これを利用して高速な処理を行っています。e^xは

$$e^x = 2^{x\log_2 e}$$
$$= 2^{[x\log_2 e]}\ 2^{x\log_2 e-[x\log_2 e]}$$

と変形することができるので、　$2^{x\log_2 e-[x\log_2 e]}$　の部分だけ近似多項式から求め、$2^{[x\log_2 e]}$ のところはあとから別に掛け合わせています。ただし、［X］は、ＩＮＴ（X）を意味するものとします。

実際のルーチンでは、まず、引数を　$X = X \times \log_2 e$　と変換した後、引数値の範囲のチェックを行います。これは次のようなものです。

（１）　$|X| \geqq 2^7$のとき

Xが正であればオーバーフローとなります。Xが負であれば結果は０となります。

（２）　$|X| < 2^{-25}$のとき

結果は０となります。

（３）　$[X]=127$ のとき

（１）と同じです。

これらのチェックをパスしたら、前述の処理を行います。あらかじめ、x＝X－［X］によって $0 \leqq x < 1$ としてから近似多項式を利用します。近似多項式：P（x）＝2 xは次のとおりです。

$$P(x)=C_0+C_1x+C_2x^2+C_3x^3+C_4x^4+C_5x^5+C_6x^6$$

ただし、

$C_0=1.00000$

$C_1=0.693147$

$C_2=0.240227$

$C_3=0.0554966$

$C_4=0.00965065$

$C_5=0.00127101$

$C_6=0.000207456$

倍精度実数型ＤＥＸＰ：９４Ｆ０Ｈ［ＳＲ：６ＤＣ２Ｈ］

領　域　９４Ｆ０Ｈ～９５６ＤＨ［ＳＲ：６ＤＣ２Ｈ～６Ｅ３ＦＨ］

機　能　倍精度実数型の指数関数

破　壊　A，F，B，C，D，E，H，L

入　力　ＦＡＣ：引数

出　力　ＦＡＣ：演算結果

ワーク
９３６ＦＨ［ＳＲ：６Ｃ４１Ｈ］：多項式係数領域（1）
９３８８Ｈ［ＳＲ：６Ｃ５ＡＨ］：多項式係数領域（2）
９３８９Ｈ［ＳＲ：６Ｃ５ＢＨ］：定数 1.0
９４２３Ｈ［ＳＲ：６ＣＦ５Ｈ］：定数 0.5
９４４３Ｈ［ＳＲ：６Ｄ１５Ｈ］：定数 $\log_2 e$
ＥＣ３ＤＨ（ＦＡＣ）：ＦＡＣの先頭アドレス
ＥＣ４５Ｈ（ＦＡＣＴＭＰ）：ＦＡＣの一時退避用
ＥＣ４ＡＨ（ＦＡＣ２）：ＦＡＣ２の先頭アドレス

ルーチン
２０ＡＢＨ（ＮＥＧ）：ＦＡＣの符号の反転
２４２４Ｈ（ＤＡＤＤ）：ＦＡＣ←ＦＡＣ＋ＦＡＣ２
２５５３Ｈ（ＤＭＵＬＴ）：ＦＡＣ←ＦＡＣ×ＦＡＣ２
２６２９Ｈ（ＤＤＩＶ）：ＦＡＣ←ＦＡＣ／ＦＡＣ２
２８５３Ｈ：オーバーフローの処理
９２４７Ｈ［ＳＲ：６Ｂ１９Ｈ］：ＦＡＣ←ＦＡＣ＋（ＨＬ）
９２４ＤＨ［ＳＲ：６Ｂ１ＦＨ］：ＦＡＣ←ＦＡＣ－（ＨＬ）

９２５３Ｈ［ＳＲ：６Ｂ２５Ｈ］：ＦＡＣ←ＦＡＣ×（ＨＬ）
９２５ＦＨ［ＳＲ：６Ｂ３１Ｈ］：ＦＡＣを整数化する
９２７３Ｈ［ＳＲ：６Ｂ４５Ｈ］：ＦＡＣ２←（ＨＬ）
９２７８Ｈ［ＳＲ：６Ｂ４ＡＨ］：ＦＡＣ←（ＨＬ）
９２８９Ｈ［ＳＲ：６Ｂ５ＢＨ］：ＦＡＣ２←ＦＡＣ
９２９ＢＨ［ＳＲ：６Ｂ６ＤＨ］（ＤＰＯＬＹＸ）：倍精度型多項式計算【２】
９２Ｂ０Ｈ［ＳＲ：６Ｂ８２Ｈ］（ＤＰＯＬＹ）：倍精度型多項式計算【１】
９２Ｄ９Ｈ［ＳＲ：６ＢＡＢＨ］：ＦＡＣをプッシュする
９２Ｅ９Ｈ［ＳＲ：６ＢＢＢＨ］：ＦＡＣ２にポップする
９２ＥＥＨ［ＳＲ：６ＢＣ０Ｈ］：ＦＡＣにポップする
９２ＦＥＨ［ＳＲ：６ＢＤ０Ｈ］：ＦＡＣの符号をＡＦレジスタに入れる
９３０ＦＨ［ＳＲ：６ＢＥ１Ｈ］：ＦＡＣ←ＦＡＣ×ＦＡＣ
９３１５Ｈ［ＳＲ：６ＢＥ７Ｈ］：ＦＡＣ２にポップした後、スタック内の値とＦＡＣの値とを交換する
９５６２Ｈ［ＳＲ：６Ｅ３４Ｈ］：結果を1.0とする
９５６８Ｈ［ＳＲ：６Ｅ３ＡＨ］：オーバーフローの場合の分岐

引数を$X=\log_2 e \times X$としたあと、次のチェックをします。

（１）　Ｘ≧２＾７のとき
　　Overflowの処理をする。
（２）　Ｘ＜２＾－５７のとき
　　結果を０とする。
（３）　Ｘの指数部が127のとき
　　Overflowの処理をする。

これらをパスしたら、Ｘを指数部と仮数部に分解します。それぞれをＹ、Ｚとすると、結果は次のようになります（ここでは有理関数近似を用いております）。

$$\frac{P(X)+Q(X)}{P(X)-Q(X)}\times 2^{Z}$$

$$P(x)=C_0+C_1x^2+C_2x^4$$

$C_0=4368.21162727559$

$C_1=233.1842114275816$

$C_2=1.000000000000000$

$$Q(x)=C_0x+C_1x^3+C_2x^5$$

C_0＝1513.9067990534

C_1＝　20.2020656512893

C_2＝　0.02302334775375023

4－3－7　対数関数

単精度実数型ＬＯＧ：ＩＦＩ０Ｈ

領　域	１Ｆ１０Ｈ～１Ｆ５２Ｈ
機　能	単精度実数型の対数関数を求める
破　壊	A，F，B，C，D，E，H，L
入　力	ＦＡＣ：引数
出　力	ＦＡＣ：関数の演算結果
ワーク	ＥＣ３ＤＨ（ＦＡＣ）：ＦＡＣの先頭アドレス １ＥＥＥＨ：多項式係数領域（１）の先頭アドレス １ＥＦＦＨ：多項式係数領域（２）の先頭アドレス
ルーチン	００２８Ｈ：ＦＡＣの先頭アドレス ０Ｂ０６Ｈ：Illegal function call エントリ １ＤＥ９Ｈ（ＦＡＤＤ）　：単精度実数型の加算 １Ｆ１８Ｈ：ＢＣＤＥレジスタにlog２の値を入れる １Ｆ２０Ｈ：有理式の計算 １Ｆ５３Ｈ（ＦＭＵＬＴ）：単精度実数型の乗算 １ＦＢ７Ｈ（ＦＤＩＶ）　：単精度実数型の除算 ２０８ＤＨ：Ａレジスタの値をＦＡＣに入れる ２０ＣＤＨ：ＦＡＣの値をスタックに積む ２０ＤＤＨ：ＢＣＤＥレジスタの値をＦＡＣに積む ２０Ｅ８Ｈ：ＦＡＣの値をＢＣＤＥレジスタにコピーする ２ＥＥ７Ｈ（ＰＯＬＹ）　：多項式の計算【１】

コンピュータが扱う数値の内部形式は、仮数部をａ；指数部をｎとすると、$a\times 2^n$ となっていますので、

$$\log n\times 2^n=\log a+n\log 2$$
$$=(\log_2 a+n)\log 2$$

と考えることができます。したがって、$\log_2 a$を求めることがこの処理系でのポイントとなります。これは次の有理関数から求めています。

$$\frac{P(X)}{Q(X)}$$

$$P(x) = C_0 + C_1 x + C_2 x^2 + C_3 x^3$$

$C_0 = -2.05467$

$C_1 = -8.86266$

$C_2 = 6.10585$

$C_3 = 4.81148$

$$Q(x) = C_0 + C_1 x + C_2 x^2 + C_3 x^3$$

$C_0 = 0.353553$

$C_1 = 4.54517$

$C_2 = 6.42784$

$C_3 = 1.00000$

倍精度実数型ＤＬＯＧ：９５６ＦＨ［ＳＲ：６Ｅ４１Ｈ］

領　域	９５６ＦＨ～９５Ｃ２Ｈ［ＳＲ：６Ｅ４１Ｈ～６Ｅ９４Ｈ］
機　能	倍精度実数型の対数関数を求める
破　壊	A，F，B，C，D，E，H，L
入　力	ＦＡＣ：引数
出　力	ＦＡＣ：演算結果
ワーク	９３８９Ｈ［ＳＲ：６Ｃ５ＢＨ］：定数 1.0
	９３Ａ１Ｈ［ＳＲ：６Ｃ７３Ｈ］：多項式定数領域
	９４３ＢＨ［ＳＲ：６Ｄ０ＤＨ］：定数$\log_e 2$
	９４４ＢＨ［ＳＲ：６Ｄ１ＤＨ］：定数$\log_e \sqrt{2}$
	９４５３Ｈ［ＳＲ：６Ｄ２５Ｈ］：定数－2.0
	９４６ＢＨ［ＳＲ：６Ｄ３ＤＨ］：定数$\sqrt{2}$
	ＥＣ３ＤＨ：ＦＡＣの先頭アドレス
	ＥＣ５２Ｈ（ＦＢＵＦＦＲ）：ＦＡＣの一時退避用
ルーチン	０Ｂ０６Ｈ：Illegal function call エントリ
	２０８ＤＨ：Ａレジスタの値を単精度実数型ＦＡＣに入れる
	２６２９Ｈ（ＤＤＩＶ）：ＦＡＣ←ＦＡＣ／ＦＡＣ２

９２４７Ｈ［ＳＲ：６Ｂ１９Ｈ］：ＦＡＣ←ＦＡＣ＋（ＨＬ）
９２４ＤＨ［ＳＲ：６Ｂ１ＦＨ］：ＦＡＣ←ＦＡＣ－（ＨＬ）
９２５３Ｈ［ＳＲ：６Ｂ２５Ｈ］：ＦＡＣ←ＦＡＣ×（ＨＬ）
９２７８Ｈ［ＳＲ：６Ｂ４ＡＨ］：ＦＡＣ←（ＨＬ）
９２７ＤＨ［ＳＲ：６Ｂ４ＦＨ］：（ＨＬ）←ＦＡＣ
９２８９Ｈ［ＳＲ：６Ｂ５ＢＨ］：ＦＡＣ２←ＦＡＣ
９２９ＢＨ［ＳＲ：６Ｂ６ＤＨ］（ＤＰＯＬＹＸ）：倍精度数型多項式計算【２】
９２ＦＥＨ［ＳＲ：６ＢＤ０Ｈ］：ＦＡＣの符号をＡＦレジスタに入れる

最初にチェックを行い、$X \leqq 0$　のとき、Illegal function call エラーとします。その後、Ｘの仮数部に対して次の変換をします。

$$X = \frac{X - 1/\sqrt{2}}{X + 1/\sqrt{2}}$$

それから近似多項式を計算します。

$$P(x) = C_0 x + C_1 x^3 + C_2 x^5 + C_3 x^7 + C_4 x^9 + C_5 x^{11} + C_6 x^{13}$$

$C_0 = 2.0000000000000000$
$C_1 = 0.6666666666633661$
$C_2 = 0.4555555512060454$
$C_3 = 0.2857140915904889$
$C_4 = 0.222238233327410$
$C_5 = 0.181113627967000$
$C_6 = 0.1694821248800000$

これも有理関数近似の一種です。なお、単精度実数型の場合は初めから分数の形をしていますが、これは倍精度実数型と同様の近似式にＸの変換式を代入して展開したものです。

4－3－8　べき乗

単精度実数型ＰＷＲ：２Ｅ１５Ｈ

領　域	２Ｅ１５～２Ｅ６ＤＨ
機　能	単精度実数型のべき乗を求める
破　壊	A，F，B，C，D，E，H，L
入　力	ＦＡＣ　：引数１（指数） ＢＣＤＥレジスタ：引数２（底）
出　力	ＦＡＣ　：演算結果
ワーク	ＥＣ３ＤＨ（ＦＡＣ）　：ＦＡＣの先頭アドレス ＥＣ４６Ｈ（ＦＬＧＯＶＣ）：オーバーフローフラグ
ルーチン	００２８Ｈ　：ＦＡＣの符号を求める １ＤＤ４Ｈ　：ＦＬＧＯＶＣを００Ｈにする １Ｅ４ＦＨ　：ＦＡＣに０をセットする １Ｆ１０Ｈ（ＬＯＧ）　：単精度実数型の対数関数 １Ｆ５３Ｈ（ＦＭＵＬＴ）　：単精度実数型の乗算 ２０ＡＢＨ（ＮＥＧ）　：ＦＡＣの符号の反転 ２０Ｅ８Ｈ　：ＦＡＣの値をＢＣＤＥレジスタに入れる ２１３４Ｈ（ＦＣＯＭＰ）　：単精度型の比較（ＢＣＤＥレジスタ－ＦＡＣ） ２２９ＦＨ　：ＦＡＣの整数化 ２８４ＣＨ　：Division by zero エントリ ２Ｅ００Ｈ：ＮＥＧルーチン（符号反転）エントリアドレスをスタックに積む（この処理系からリターンするときにＮＥＧを実行するように） ２Ｅ６ＥＨ（ＥＸＰ）　：単精度実数型の指数関数 ＥＤＣ６Ｈ：フック（ＲＥＴ）

まず、次のチェックを行います。『Ｘ＾Ｙ』において、

（１）　Ｙ＝０のとき
結果を1.0とする（すなわち、ゼロのゼロ乗は『１』となる）

（２）　Ｘ＝０で、かつ、Ｙが負の数のとき
Divsion by zeroエラー となる

（３）　Ｘ＝０で、（１）・（２）のいずれにも該当しないとき

結果を０とする

（４）　Ｘ＜０でＹが奇数のとき

ＮＥＧルーチンのアドレスをスタックに積んで最終結果の符号を反転する

（５）　Ｘ＜０でＹが負の整数のとき［（４）のケースも包含］

Ｘを正にする

以上を確認の上、Ｘ＾Ｙ＝ exp（Ｙ× log（Ｘ））という関係を用いて演算結果を計算します。ところで、（４）・（５）からわかるように、Ｘ＜０でＹが負の非整数の場合には、Ｘは負のままです。この場合には、log（Ｘ）を呼び出すときに、Illegal function call エラーが発生します。

倍精度実数型ＤＰＷＲ：９５Ｃ３Ｈ［ＳＲ：６Ｅ９５Ｈ］

領　域	９５Ｃ３Ｈ～９６１４Ｈ［ＳＲ：６Ｅ９５Ｈ～６ＥＥ６Ｈ］	
機　能	倍精度実数型のべき乗を求める	
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ	
入　力	ＦＡＣ　：引数１（底）	
	ＦＡＣ２：引数２（指数）	
出　力	ＦＡＣ：演算結果	
ワーク	９３８９Ｈ［ＳＲ：６Ｃ５ＢＨ］	：定数 1.0
	ＥＣ４ＡＨ（ＦＡＣ２）	：ＦＡＣ２の先頭アドレス
	ＥＣ５２Ｈ（ＦＢＵＦＦＲ）	：計算結果の一時退避用
	ＥＣ５ＡＨ（ＦＢＵＦＲ２）	：計算結果の一時退避用
ルーチン	０Ｂ０６Ｈ	：Illegal function call エラーエントリ
	２０ＡＢＨ（ＮＥＧ）	：ＦＡＣの符号を反転する
	２８４ＣＨ	：オーバーフローの処理
	９２５３Ｈ［ＳＲ：６Ｂ２５Ｈ］	：ＦＡＣ←ＦＡＣ＊（ＨＬ）
	９２５９Ｈ［ＳＲ：６Ｂ２ＢＨ］	：ＦＡＣ－(ＨＬ)の符号をＡＦレジスタに代入
	９２５ＦＨ［ＳＲ：６Ｂ３１Ｈ］	：ＦＡＣを整数化してＡレジスタに代入
	９２７８Ｈ［ＳＲ：６Ｂ４ＡＨ］	：ＦＡＣ←（ＨＬ）
	９２７ＤＨ［ＳＲ：６Ｂ４ＦＨ］	：（ＨＬ）←ＦＡＣ
	９２８ＥＨ［ＳＲ：６Ｂ６０Ｈ］	：ＦＡＣ←ＦＡＣ２
	９２９６Ｈ［ＳＲ：６Ｂ６８Ｈ］	：ＮＥＧルーチンエントリアドレスをスタックに積む（ＤＰＷＲルーチンからリターンするときにＮＥＧを実行するため）

９２ＦＥＨ［ＳＲ：６ＢＤ０Ｈ］：ＦＡＣの符号をＡＦレジスタに入れる
９４Ｆ０Ｈ［ＳＲ：６ＤＣ２Ｈ］（ＤＥＸＰ）：倍精度実数型の指数関数
９５６ＦＨ［ＳＲ：６Ｅ４１Ｈ］（ＤＬＯＧ）：倍精度実数型の対数関数
９６０ＣＨ［ＳＲ：６ＥＤＥＨ］：引数１（底）が０のときの分岐

まず次のチェックをします。『Ｘ＾Ｙ』において、

（１）　Ｙ＝０のとき
結果を1.0とする
（２）　Ｘ＝０　かつ　Ｙ≧０のとき
結果を０とする
（３）　Ｘ＝０　かつ　Ｙ＜０　のとき
Divsion by zero となる
（４）　Ｘ＜０　のとき
Ｘ＝－Ｘ（＞０）とする
（５）　最初のＸが負の非整数だったとき
Illegal function call となる
（６）　最初のＸが負の奇数だったとき
このルーチンを出るときに結果を反転するように、ＮＥＧルーチンのエントリアドレスをスタックに積む

以上のチェックを終えたら、Ｘ＾Ｙ＝exp（Ｙ×log（Ｘ））の関係を利用して演算結果を求めます。

４－３－９　正弦関数

単精度実数型ＳＩＮ：２Ｆ９１Ｈ

領　域	２Ｆ９１Ｈ～３００６Ｈ	
ワーク	単精度実数型の正弦関数を求める	
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ	
入　力	ＦＡＣ：引数	
出　力	ＦＡＣ：演算結果	
ワーク	３０１７Ｈ	：単精度正弦関数用係数領域
	ＥＣ３Ｄ（ＦＡＣ）	：ＦＡＣの先頭アドレス
ルーチン	００２８Ｈ	：ＦＡＣの符号を調べる（フラグが変化する）

１ＤＥ６Ｈ（ＦＳＵＢ）	：単精度実数型の減算
１ＤＥ９Ｈ（ＦＡＤＤ）	：単精度実数型の加算
１Ｆ５３Ｈ（ＦＭＵＬＴ）	：単精度実数型の乗算
２０ＡＢ（ＮＥＧ）	：ＦＡＣの符号を反転する
２０ＣＤＨ	：ＦＡＣをスタックに積む
２１３４Ｈ（ＦＣＯＭＰ）	：単精度実数型の比較（結果が負となるのはＢＣＤＥレジスタ＞ＦＡＣのとき）
２２９ＦＨ	：ＦＡＣを整数化する
２ＥＤ８Ｈ（ＰＯＬＹＸ）	：多項式計算ルーチン【２】
２ＦＡ７Ｈ	：ＢＣＤＥレジスタに１／２π＝0.159155 を代入
２ＦＢＢＨ	：ＦＡＣに＋0.25 を代入
２ＦＣ７Ｈ	：ＦＡＣに＋0.5 を代入
２ＦＤ０Ｈ	：ＦＡＣに－0.5 を代入
２ＦＤＤＨ	：ＦＡＣに－0.25 を代入

まず次のチェックを行います。

（１）　Ｘの指数部が－９未満（Ｘ＜9.765623 Ｅ－3）のとき
　結果はＸとする（ｘ≒０　ならば　sin ｘ≒ｘ）
（２）　Ｘ＜０のとき
　sin（ｘ）＝－ sin（－ｘ）という関係を利用して、引数の符号を正として以下の処理を行い、最終結果の符号を反転（ＮＥＧルーチンの利用）する
（３）　sin（２ｎπ＋ｘ）＝sin ｘという関係を利用し、Ｘの範囲を０≦Ｘ＜２πに収める
（４）　３π／２≦ｘ＜２πのとき
　sin ｘ＝ sin（ｘ－２π）であるので、Ｘ＝Ｘ－２πとする
（５）　π／２≦ｘ＜３π／２のとき
　sin ｘ＝ sin（π－ｘ）であるので、Ｘ＝π－Ｘとする
（６）　この時点で、Ｘ＜０のとき
　（２）と全く同じことをする

以上により、引数Ｘの値の範囲は　０≦Ｘ＜π／２　となります。ただし、実際には（１）と（２）の間で、Ｘ＝Ｘ／２πという操作を行っています。そのため、ＦＡＣ上でのＸの値は、０≦Ｘ＜0.25 です［（３）のチェックもルーチン内では 1.5≦Ｘ＜２を調べている］。

ここで使用する近似多項式は次のようなものです。

$$P(X) = C_0 x + C_1 x^3 + C_2 x^5 + C_3 x^7 + C_4 x^9$$

$C_0=\ \ 6.28319$
$C_1=-41.3471$
$C_2=\ \ 64.1001$
$C_3=-76.5750$
$C_4=\ \ 39.7109$

倍精度実数型ＤＳＩＮ：９６１５Ｈ［ＳＲ：６ＥＥ７Ｈ］

領　域	９６１５Ｈ～９６４ＦＨ［ＳＲ：６ＥＥ７Ｈ～６Ｆ２１Ｈ］
機　能	倍精度実数型の正弦関数
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＦＡＣ：引数
出　力	ＦＡＣ：演算結果
ワーク	９３ＤＡＨ［ＳＲ：６ＣＡＣＨ］：多項式係数領域
	９４２３Ｈ［ＳＲ：６ＣＦ５Ｈ］：定数0.5
	９４２ＢＨ［ＳＲ：６ＣＦＤＨ］：定数１／２π
	９４３３Ｈ［ＳＲ：６Ｄ０５Ｈ］：定数0.25
ルーチン	２０ＡＢＨ（ＮＥＧ）：ＦＡＣの符号を反転する
	２４２４Ｈ：ＦＡＣ←ＦＡＣ＋ＦＡＣ２
	９２４７Ｈ［ＳＲ：６Ｂ１９Ｈ］：ＦＡＣ←ＦＡＣ－（ＨＬ）
	９２４ＤＨ［ＳＲ：６Ｂ１ＦＨ］：ＦＡＣ←ＦＡＣ－（ＨＬ）
	９２５３Ｈ［ＳＲ：６Ｂ２５Ｈ］：ＦＡＣ←ＦＡＣ×（ＨＬ）
	９２５ＦＨ［ＳＲ：６Ｂ３１Ｈ］：ＦＡＣを整数化する
	９２９ＢＨ［ＳＲ：６Ｂ６ＤＨ］（ＤＰＯＬＹＸ）：倍精度型の多項式の計算
	９２Ｄ９Ｈ［ＳＲ：６ＢＡＢＨ］：ＦＡＣをスタックに積む
	９２Ｆ９Ｈ［ＳＲ：６ＢＢＢＨ］：ＦＡＣ２にポップする
	９２ＦＥＨ［ＳＲ：６ＢＤ０Ｈ］：ＦＡＣの符号をＡＦレジスタに代入

最初に、Ｘを２πで割り、その小数部だけを取り出したあと、その値から0.25を引いたものを新たなＸとして次のチェックをします（このときの符号をＳとします）。

（1）　Ｓ≧０のとき
Ｘ＝Ｘ－0.5とする；ただし、この値が正のときは、さらにこの値を反転する

ここでＸ＝Ｘ＋0.25とします。

（２）　再びＳ≧０のとき
　　Ｘの値を反転する

ここで次の近似多項式を計算します。

$$P(X)=\ C_0x+C_1x^3+C_2x^5+C_3x^7+C_4x^9+C^5x_{11}+C_6x^{13}+C_7x^{15}+C_8x^{17}$$

$C_0=\ \ 6.283185307179587$
$C_1=-41.34170224039976$
$C_2=\ \ 81.60524927607341$
$C_3=-76.70585975279850$
$C_4=\ \ 42.05869392378517$
$C_5=-15.09464162050745$
$C_6=\ \ 3.819927050429582$
$C_7=-\ 0.7177235796522383$
$C_8=\ \ 0.100856328659316$

４－３－１０　余弦関数

単精度実数型ＣＯＳ：２Ｆ８ＢＨ

領　域	２Ｆ８ＢＨ〜２Ｆ９０Ｈ
機　能	単精度実数型の余弦関数を求める
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＦＡＣ：引数
出　力	ＦＡＣ：演算結果
ワーク	３００ＦＨ：π／２の値（４バイト）
ルーチン	１ＤＤＥＨ：ＦＡＣからＨＬレジスタの指す４バイトの値を引く ２Ｆ９１Ｈ（ＳＩＮ）：単精度実数型の正弦関数

ＣＯＳ（Ｘ）＝ＳＩＮ（Ｘ＋π／２）より計算します。

倍精度実数型ＤＣＯＳ：９４Ｅ７Ｈ［ＳＲ：６ＤＢ９Ｈ］

領　域	９４Ｅ７〜９４ＥＦＨ［ＳＲ：６ＤＢ９Ｈ〜６ＤＣ１Ｈ］

機　　能	倍精度実数型の余弦関数
破　　壊	A，F，B，C，D，E，H，L
入　　力	FAC：引数
出　　力	FAC：演算結果
ワ ー ク	945BH［SR：6D2DH］：定数π／2
ルーチン	9247H［SR：6B19H］：FAC←FAC＋（HL） 9615H［SR：6EE7H］（DSIN）：単精度実数型の正弦関数

COS（X）＝SIN（X＋π／2）より計算します。

4－3－11　正接関数

単精度実数型TAN：302CH

領　　域	302CH～3040H
機　　能	単精度実数型の正接関数を求める
破　　壊	A，F，B，C，D，E，H，L
入　　力	FAC：引数
出　　力	FAC：演算結果
ワ ー ク	なし
ルーチン	1F85H（FDIV）：単精度実数型の除算 20CDH：FACをスタックに積む 20DDH：BCDEレジスタの値をFACに代入 2F8BH（COS）：単精度実数型の余弦関数 2F91H（SIN）：単精度実数型の正弦関数

TAN（X）＝SIN（X）／COS（X）より求めます。

倍精度実数型DTAN：9691H［SR：6F63H］

領　　域	9691H～96A2H［SR：6F63H～6F74H］
ワ ー ク	倍精度実数型の正接関数
破　　壊	A，F，B，C，D，E，H，L
入　　力	FAC：引数

出　　力	ＦＡＣ：演算結果
ワーク	なし
ルーチン	２６２９Ｈ（ＤＤＩＶ）：ＦＡＣ←ＦＡＣ／ＦＡＣ２ ９２Ｄ９Ｈ［ＳＲ：６ＢＡＢＨ］：ＦＡＣをプッシュする ９２Ｅ９Ｈ［ＳＲ：６ＢＢＢＨ］：ＦＡＣ２にポップする ９３１５Ｈ［ＳＲ：６ＢＥ７Ｈ］：スタック上の値とＦＡＣの値を交換する ９４Ｅ７Ｈ［ＳＲ：６ＤＢ９Ｈ］（ＤＣＯＳ）：倍精度実数型の余弦関数 ９６１５Ｈ［ＳＲ：６ＥＥ７Ｈ］（ＤＳＩＮ）：倍精度実数型の正弦関数

ＴＡＮ（Ｘ）＝ＳＩＮ（Ｘ）／ＣＯＳ（Ｘ）より求めます。

４－３－１２　平方根

単精度実数型ＳＱＲ：２Ｅ０５Ｈ

領　　域	２Ｅ０５Ｈ～２Ｅ０９Ｈ
機　　能	単精度実数型の平方根を求める
破　　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　　力	ＦＡＣ：引数
出　　力	ＦＡＣ：演算結果
ワーク	２Ｄ５ＡＨ：定数 0.5（４バイト）
ルーチン	２０ＣＤＨ：ＦＡＣをスタックにプッシュする ２０ＤＡＨ：ＦＡＣにＨＬレジスタから始まる４バイトの値を代入 ２Ｅ１３Ｈ：スタックからＢＣＤＥレジスタにポップした後、単精度実数型のべき乗ルーチンを呼ぶ

ＳＱＲ（Ｘ）＝Ｘ＾0.5 より計算します。

倍精度実数型ＤＳＱＲ：９６５０Ｈ［ＳＲ：６Ｆ２２Ｈ］

領　　域	９６５０Ｈ～９６９０Ｈ［ＳＲ：６Ｆ２２Ｈ～６Ｆ６２Ｈ］
機　　能	倍精度実数型の平方根
破　　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　　力	ＦＡＣ：引数

出　　力	ＦＡＣ：演算結果
ワ ー ク	９４２３Ｈ［ＳＲ：６ＣＦ５Ｈ］：定数0.5
	ＥＣ３ＤＨ（ＦＡＣ）　：ＦＡＣの先頭アドレス
	ＥＣ４ＡＨ（ＦＡＣ２）：ＦＡＣ２の先頭アドレス
ルーチン	０Ｂ０６Ｈ：Illegal function call エラーエントリ
	２４２４Ｈ：ＦＡＣ←ＦＡＣ＋ＦＡＣ２
	２６２９Ｈ（ＤＤＩＶ）：ＦＡＣ←ＦＡＣ／ＦＡＣ２
	９２５３Ｈ［ＳＲ：６Ｂ２５Ｈ］：ＦＡＣ←ＦＡＣ×（ＨＬ）
	９２８９Ｈ［ＳＲ：６Ｂ５ＢＨ］：ＦＡＣ２←ＦＡＣ
	９２８ＥＨ［ＳＲ：６Ｂ６０Ｈ］：ＦＡＣ←ＦＡＣ２
	９２Ｄ４Ｈ［ＳＲ：６ＢＡ６Ｈ］：ＦＡＣ２をプッシュする
	９２Ｄ９Ｈ［ＳＲ：６ＢＡＢＨ］：ＦＡＣをプッシュする
	９２Ｅ９Ｈ［ＳＲ：６ＢＢＢＨ］：ＦＡＣ２にポップする
	９２ＥＥＨ［ＳＲ：６ＢＣ０Ｈ］：ＦＡＣにポップする
	９２ＦＥＨ［ＳＲ：６ＢＤ０Ｈ］：ＦＡＣの符号をＡＦレジスタに代入

まず、次のチェックを行います。

（１）　Ｘ＝０のとき
そのままリターンする（結果は０）

（２）　Ｘ＜０のとき
Illeagl function call エラー

ここでＳＱＲ（$a\times 2^{2n}$）＝2^n×ＳＱＲ（ａ）を利用するため、Ｘの指数部を２で割ったものをスタックに詰め込み、新たなＸの指数部としてこのときの剰余を設定します。たとえば、（Ｘの指数部）＝41のとき、20をスタックに入れて（Ｘの指数部）＝１に変更します。その後、ニュートン法により解を求めます。$\sqrt{a}$ を求める場合、$y=x^2-a$ にニュートン法を適用し、

$$x_{n+1}=\frac{1}{2}\left(\frac{a}{x_n}+x_n\right)$$

という式を繰り返すことによって $\{x_n\}$ は真の値に近づきます。このルーチンでは初期値として $x_0=a$ を用いたあと、x_5 まで計算しています。この値が求められたら、このときの指数部に、先程スタックに積み込んだ値を加えて計算結果とします。

第5章

入　出　力

5

CONTENTS

5－1　ファイル

5－1－1　ファイルディスクリプタ

N_{88}－ＢＡＳＩＣでは、ファイルの概念を導入することによって、すべての入出力機器の操作を統一しています。つまり、ディスクへの出力もスクリーンへの出力も同じコマンドで行うことができるのです。このとき、各ファイルを区別する名前の役割を果たすのがファイルディスクリプタです。

ファイルディスクリプタは次のような構成になっています。

"＜デバイス名＞［＜オプション＞］［＜ファイル名＞］"

デバイス名は入出力機器の名称（種類）を表わすもので、図５－１－１のように定められて

デバイス名	機械名称	入力	出力
ＬＰＴ１： ＬＰＴ：	プリンタ	×	○
ＳＣＲＮ：	スクリーン	×	○
ＣＯＭ１： ＣＯＭ：	ＲＳ－２３２Ｃポート１	○	○
ＣＡＳ１： ＣＡＳ：	カセットテープ１（1200bps）	○	○
ＣＡＳ２：	カセットテープ２（600bps）	○	○
１：	フロッピィディスク１	○	○
２：	フロッピィディスク２	○	○
３：	フロッピィディスク３	○	○
４：	フロッピィディスク４	○	○
５：	フロッピィディスク５	○	○
６：	フロッピィディスク６	○	○
７：	フロッピィディスク７	○	○
８：	フロッピィディスク８	○	○
ＫＹＢＤ：	キーボード	○	×

図5－1－1　デバイス名

います。オプションはＲＳ−２３２Ｃをデバイスとして指定したときに各種のモードを設定するもので、次のようになっています（図５−１−２参照）。

［＜パリティ＞［＜ビット長＞［＜ストップビット＞［＜Ｘパラメータ＞［＜Ｓパラメータ＞］］］］］

オプションのそれぞれの意味は５−６−１節を参照してください。

ファイル名はこのプログラムファイルやデータファイルに付ける名前で、ユーザが自由に決めることのできるものです。これは次のような形式となっています。

＜ファイル名＞［.］［＜拡張子＞］

＜ファイル名＞、＜拡張子＞はそれぞれ６文字、３文字まで有効とされ、それに満たない場合には空白が補充されます。

パリティ：パリティチェックの指示
- 偶数（even）パリティ------------------------------------E
- 奇数(odd)パリティ--------------------------------------O
- パリティチェックなし(non)------------------------------N

ビット長：データ１文字の長さ
- ７ビット---7
- ８ビット---8

ストップビット：ストップビットの長さ
- １ビット---1
- 1.5ビット--2
- ２ビット---3

Ｘパラメータ：Ｘパラメータ有効、無効の切り換え
- Ｘパラメータ有効---------------------------------------X
- Ｘパラメータ無効---------------------------------------N

Ｓパラメータ：Ｓパラメータ有効、無効の切り換え
- Ｓパラメータ有効---------------------------------------S
- Ｓパラメータ無効---------------------------------------N

※　すべて大文字でこの順番に書く。途中を省略して後ろのパラメータを書きたいときは、該当部分をスペースにする。

(例)------Ｅ８１␣Ｓ

無指定の場合は本体のディップスイッチＳＷ２の１〜５による。

図５−１−２　オプション

5－1－2　ドライブテーブルとファイルバッファ

ドライブポインタからファイルバッファまでについてここで説明します（両者のメモリロケーションは第4章の図4－1－1参照のこと）。

まず、ドライブバッファ（ドライブテーブル）というのは、接続されている各ドライブのステータスやＦＡＴのコピーが記録されている領域のことで、１ドライブにつき165バイト使用しています(図５－１－３)。このドライブテーブルの位置はドライブポインタから知ることができます（図５－１－４)。

次に、ファイルバッファというのは、ディスクやＲＳ－２３２Ｃが入出力を行う際に使用する緩衝領域です(図５－１－５)。これは、9バイトのワークエリアであるＦＣＢ(File Controle Block）と、それに続く256バイトの入出力バッファとで構成されます（図５－１－６)。ディスクでは、入出力の際にはセクタ単位（ＰＣ－８８０１では256バイト）でしかデータのやりとりができませんので、このようなバッファが必要になります。ＲＳ－２３２Ｃでは、データの受信の際のキューバッファとしてこの領域を使います（５－２－３節参照)。

バッファアドレス－2 ファイルへの書き込みが行われるとＦＦＨとなり、リムーブ時にＦＡＴの更新が行われる
バッファアドレス－1 ドライブの属性を示す
バッファアドレス 最後にアクセスしたトラック番号、ディスクがマウントされていないときはＦＦＨ
バッファアドレス＋1 未使用
バッファアドレス＋2 残りのクラスタ数
バッファアドレス＋3～＋163 ＦＡＴのコピー：5インチのとき160バイト、8インチのとき154バイト

図5－1－3　ドライブバッファ(ドライブテーブル)

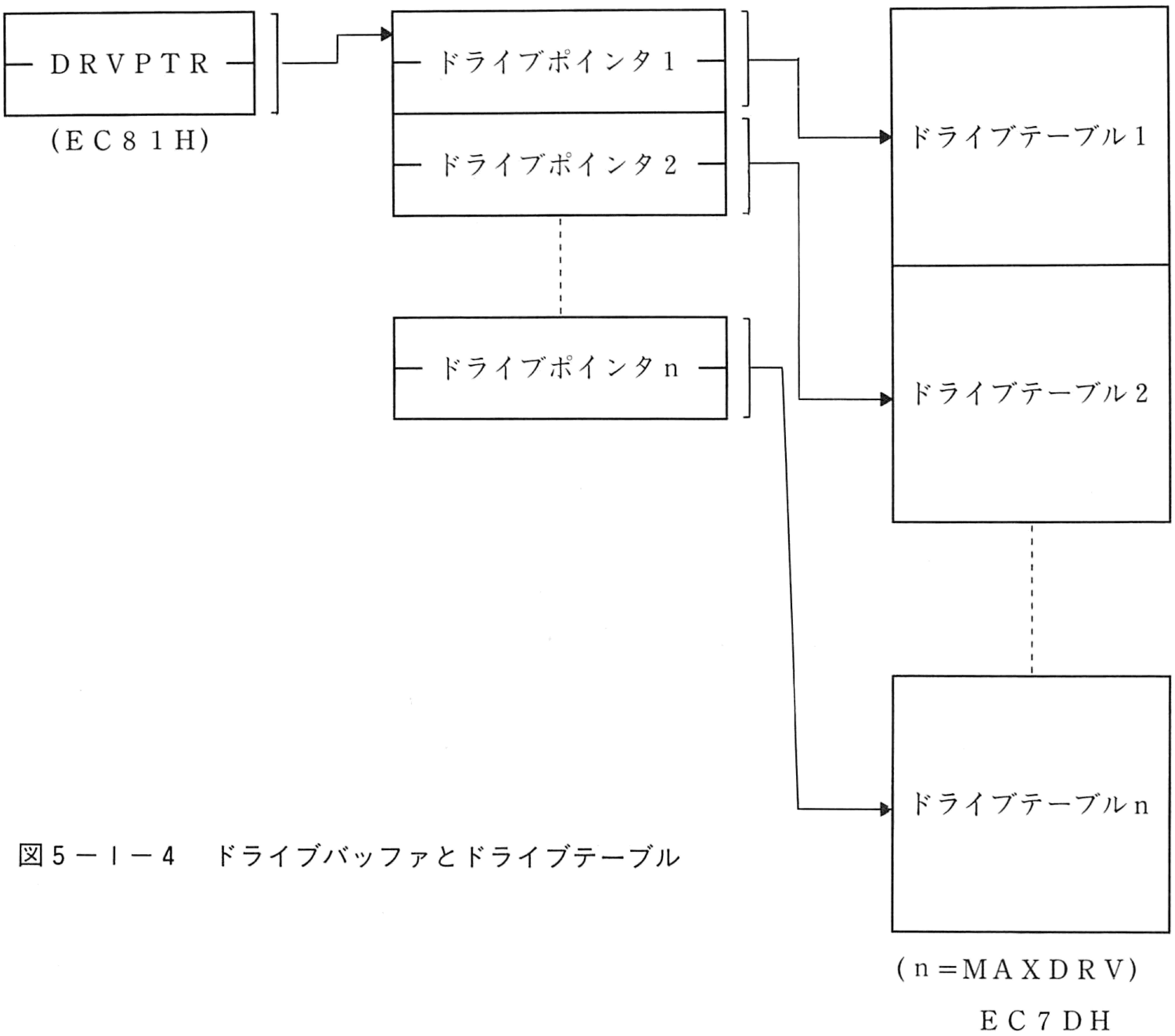


図5－Ⅰ－4　ドライブバッファとドライブテーブル

5－Ⅰ－3　ファイル関係ルーチン

ファイルバッファアドレス計算ルーチン：NAMSCN

領　域	46F8H～4721H
機　能	ファイルバッファの先頭アドレスを求める
破　壊	A，F，D，E，H，L
入　力	Aレジスタ：ファイルナンバー
出　力	HLレジスタ：ファイルバッファの先頭アドレス Cフラグ：0のときディスクファイル、1のときディスク以外のファイル Zフラグ：0のとき指定ファイルオープン、1のとき指定ファイルクローズ

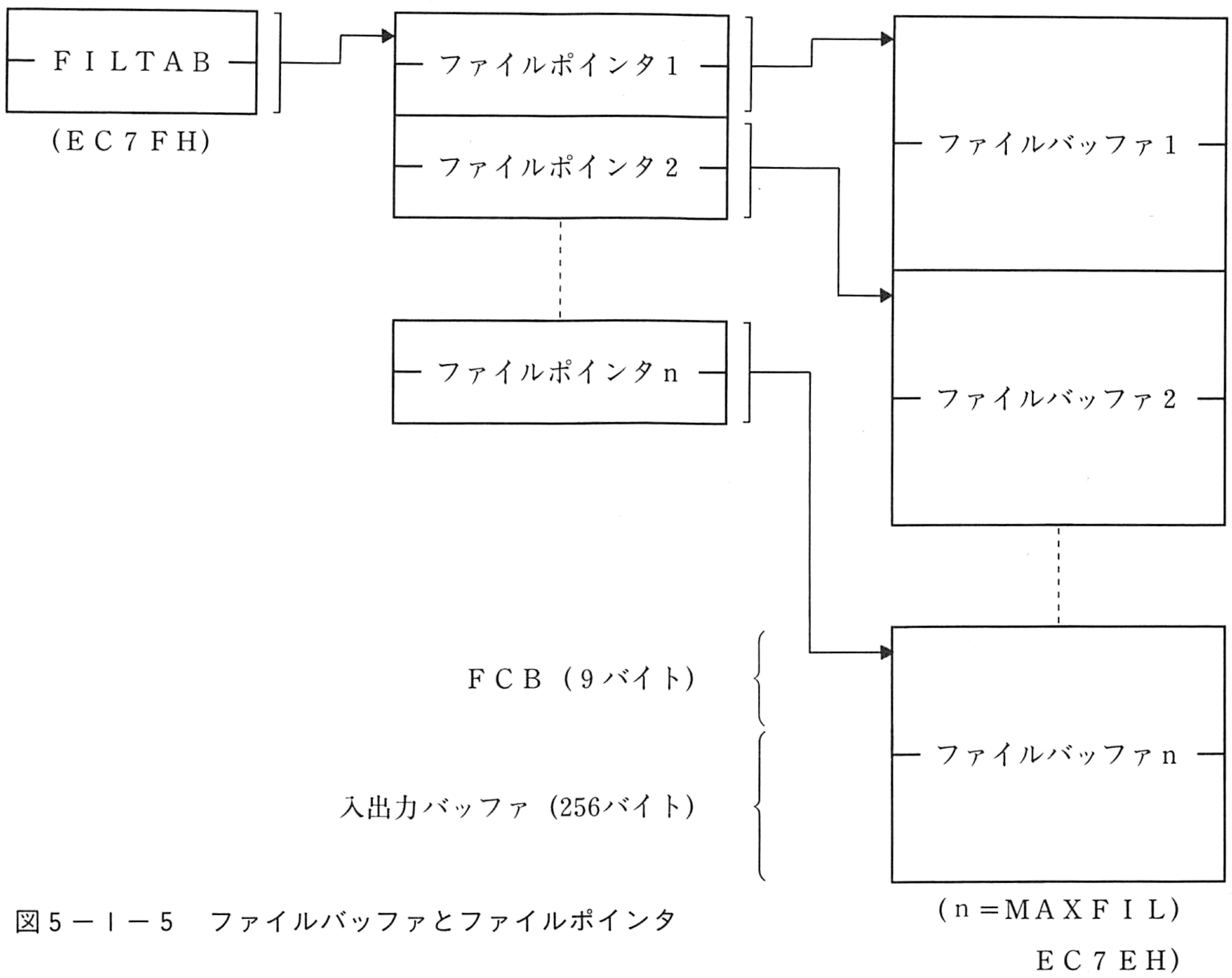
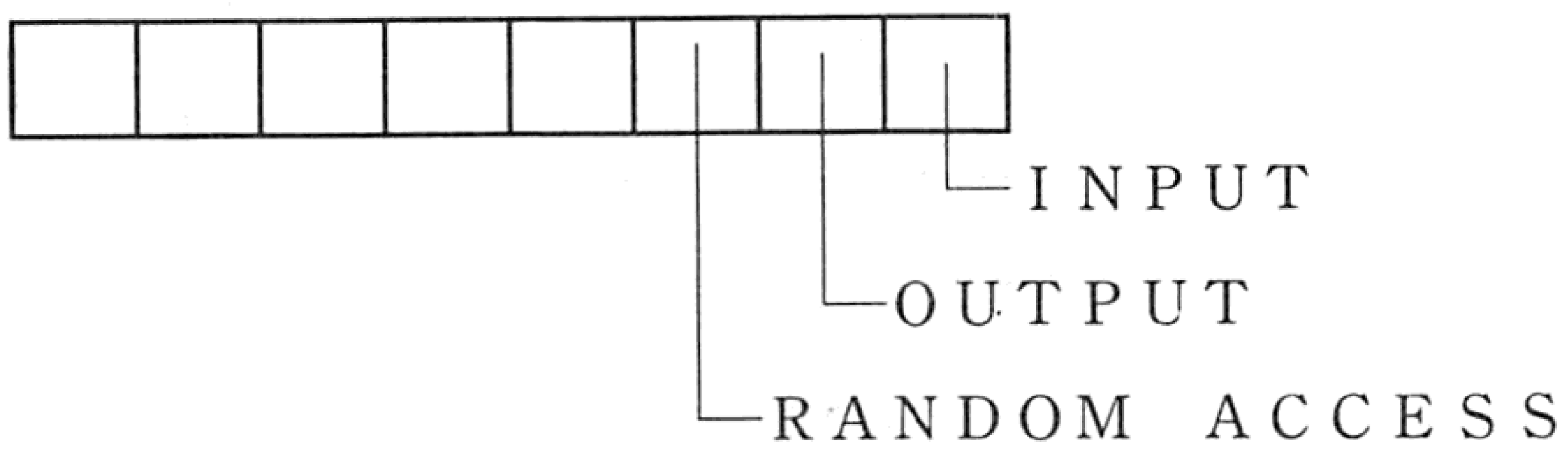


図５－１－５　ファイルバッファとファイルポインタ

ワーク		
	EC7EH	(MAXFIL)：最大ファイル数
	EC7FH	(FILTAB)：ファイルバッファナンバー０のポインタアドレス
	EC85H	(CURDRV)：カレントドライブ番号
	ECA3H	(NLONLY)：ターミナルモード用フラグ
	ECA5H	(MAXTRK)：最大トラック番号
	ECA6H	(NUMSEC)：１トラックあたりのセクタ数
	ECA7H	(TWOSUR)：ディスク両面フラグ
	ECA8H	(CLSTRK)：１トラックあたりのクラスタ数
	ECA9H	(NUMCLS)：ドライブ上の総クラスタ数
	ECAAH	(DIRTRK)：ディレクトリのあるトラック番号
	ECABH	(CLSIZ)：１クラスタあたりのセクタ数

バッファアドレス＋０：ファイルモード



１のビットがあれば、そのモードであることを示す；クローズされているときは００H

バッファアドレス＋１：先頭クラスタ番号

ファイルの格納されている先頭クラスタ番号；ディスクファイルでのみ有効

バッファアドレス＋２：カレントクラスタ番号

現在読み書きしているクラスタの番号；ディスクファイルでのみ有効

バッファアドレス＋３：カレントセクタ番号

現在読み書きしているセクタの番号；ディスクファイルでのみ有効

バッファアドレス＋４：デバイス番号

フロッピィディスク１---------------------００H
フロッピィディスク２---------------------０１H
フロッピィディスク３---------------------０２H
フロッピィディスク４---------------------０３H
フロッピィディスク５---------------------０４H
フロッピィディスク６---------------------０５H
フロッピィディスク７---------------------０６H
フロッピィディスク８---------------------０７H
スクリーン-------------------------------F８H
プリンタ---------------------------------F９H
カセットテープ１-------------------------FAH
カセットテープ２-------------------------FBH
RS－２３２Cポート１--------------------FEH
キーボード-------------------------------FFH

バッファアドレス＋５：バッファサイズ

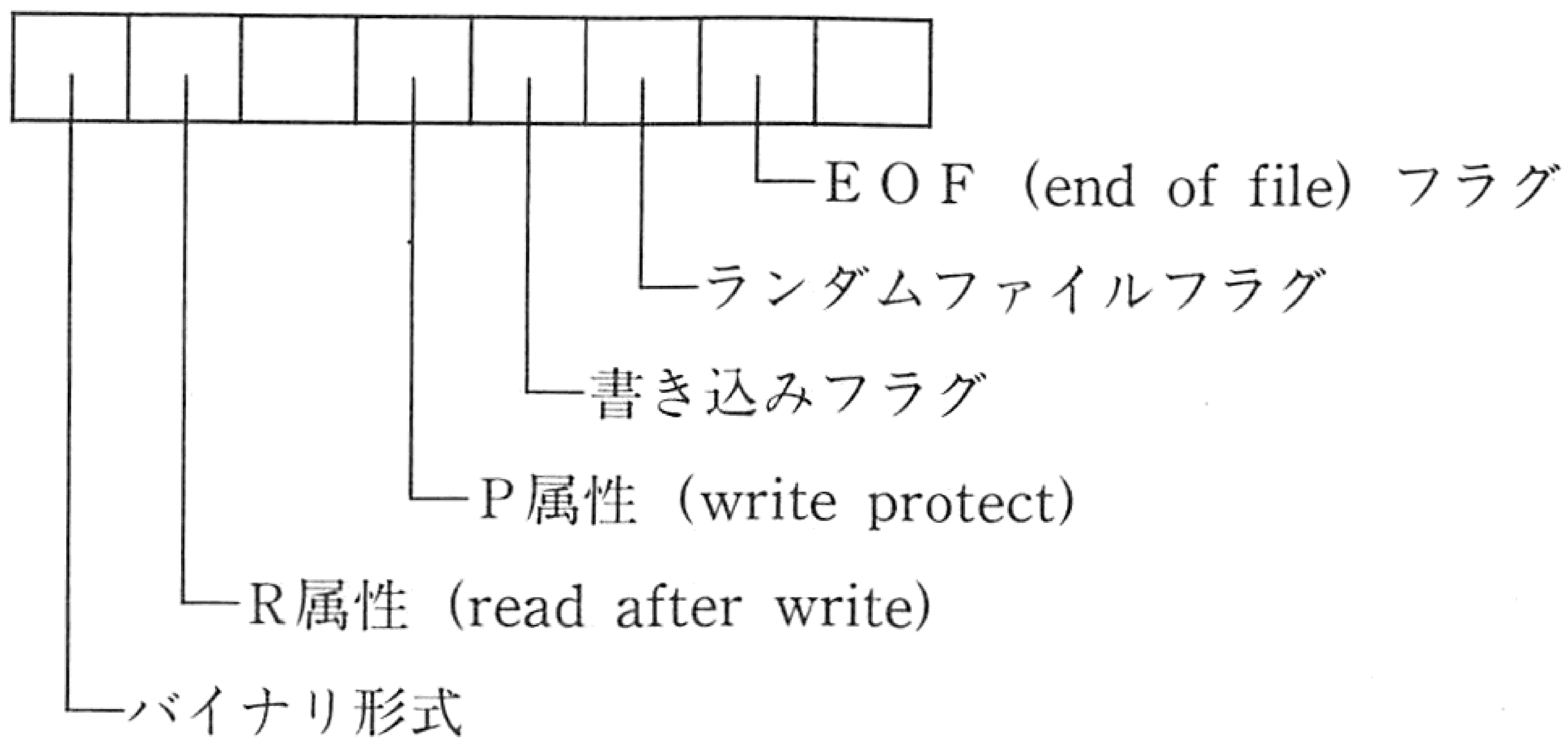
入出力バッファの大きさ；初期値は００H＝256バイト；プリンタとRS－２３２Cポートのとき有効

図５－１－６①　FCBの構成（２の１）

バッファアドレス＋６：レコードポインタ

現在読み書きしているレコードの、入出力バッファ内における位置（先頭からのオフセット）；ディスクファイルをシーケンシャルファイルとして使用するときのみ有効

バッファアドレス＋７：属性、ステータス



（１のときそれぞれ意味をもつ）

バッファアドレス＋８：データポインタ

レコード内で次にデータを読み書きする位置（レコードの先頭からのオフセット）；シーケンシャルファイルで有効

図５－１－６②　FCBの構成（２の２）

ＥＣＡＣＨ（ＦＡＴＯＮＥ）：ＦＡＴの先頭セクタ番号
ＥＣＡＤＨ（ＦＡＴＬＳＴ）：ＦＡＴの終了セクタ番号
ＥＣＡＥＨ（ＦＡＴＮＵＭ）：ＦＡＴの数
ＥＣＡＦＨ（ＤＳＫＩＮＦ）：ディスク属性（ＩＤ）が入っているセクタ番号

ルーチン

４７２４Ｈ：ディスクでない時の終了
４ＤＢ２Ｈ：Bad file number エラーエントリ
４ＤＢ５Ｈ：Internal error エントリ
ＥＤＢ１Ｈ：フック（ＪＰ　９９４ＢＨ、ＪＰ　８ＣＦ３Ｈ）；ディスク時の処理（ディスクの最大ドライブ数を越えていないことを確認し、各パラメータを設定する）

まず最初に、ファイルナンバーが最大のファイル数を越えていないことを確認します（越えていたらBad file number エラー)。それから、ファイルバッファの先頭アドレスを求めることになりますが、これはファイルポインタが指し示しています(図５－１－４参照)。ファイルナンバーｎのファイルポインタは次の式により求めることができます。

ＦＩＬＴＡＢ＋ｎ×２

ファイルバッファのアドレスが求められたらＮＬＯＮＬＹを調べ、これがＦＦＨであればここでリターンします。また、ファイルがクローズされていてＦＣＢの先頭のバイトが００Ｈであるときもすぐにリターンします。これらをパスしたならば、ＦＣＢからデバイス番号を求めてＡレジスタに代入します。ここで、デバイス番号が９以上の場合には、デバイスがディスク以外ですから、Ｃフラグをセットしてこのルーチンから戻ります。デバイスがディスクの場合には、フックによって８ＣＦ３Ｈ番地にジャンプします。ここでは、デバイス番号がＭＡＸＤＲＶを越えていないのを確認し、ディスクの各パラメータをセットします（ＣＵＲＤＲＶ，ＭＡＸＴＲＫ～ＤＳＫＩＮＦ)。それから、ＤＲＶＴＢＬとＤＲＶＰＴＲを今回指定したドライブのものに変更します。

ファイルディスクリプタ処理ルーチン：ＳＰＣＤＳＫ

領　域	４６８ＣＨ
機　能	ファイルディスクリプタ処理を行い、デバイスナンバーとファイル名を得る
破　壊	Ａ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＨＬレジスタ：ファイルディスクリプタの格納されている先頭アドレス
出　力	Ａレジスタ、Ｄレジスタ　：デバイスナンバー
	ＥＣ８ＦＨ（ＦＩＬＮＡＭ）：ファイル名（９バイト）
ワーク	ＥＣ８ＦＨ（ＦＩＬＮＡＭ）：ファイル名バッファ１
ルーチン	１１Ｄ３Ｈ：文字列式（ファイルディスクリプタ）を評価し、ＦＡＣに代入
	４６Ｃ５Ｈ：Bad file number エラーエントリ
	４６ＣＢＨ：ファイル名の“．”以前の部分が６文字に満たないとき、スペースで埋める
	４６ＤＦＨ：ファイル名の最後に００Ｈ（ＣＡＳの時)、または、２０Ｈ（ＣＡＳ以外のとき）をセット
	４Ｄ６６Ｈ：デバイス名の処理
	５６Ｃ９Ｈ：ＦＡＣが文字列であること確認した上で取り出す

最初に、ファイルディスクリプタをＦＡＣに取り込みます。次に、それが文字列であることを確かめ、ＦＡＣから取り出します（そのため、ファイルディスクリプタはダブルクォーテー

ションマーク（"）で始まり、ダブルクォーテーションマークもしくは００Ｈで終わる必要があります）。このとき文字数が０であれば、Bad file name エラーとなります。それから、ファイルディスクリプタのデバイス部を取り出します。これが省略された場合には、"ＣＡＳ１："（ＲＯＭバージョン）または"１："（ディスクバージョン）とみなされます。その後でファイルディスクリプタをＦＩＬＮＡＭに書き込みます。このとき、ＡＳＣＩＩコードにおける２０Ｈより小さい文字が含まれているとBad file name エラーとなります。また、"．"以前に６文字ないときは、不足分をスペースで補います。最後に、ファイルディスクリプタの先頭がＦＦＨでないのを確認したうえで、このルーチンを終えます。

５－１－４　キュー

キーボード、カセット、ＲＳ－２３２Ｃの入力では、バッファにキューを使用しています。キューというのは、次々と入力していったデータを古いものから順番に取り出していくバッファで、ＦＩＦＯ（First In First Out）バッファとも呼ばれます。一続きのメモリ領域の上端と下端を円形につなぎ合わせてたものと考えればよいでしょう。この中を、以前にデータを書き込んだ位置を示すポインタ（プットポインタ）と、以前にデータを取り出した位置を示すポインタ（ゲットポインタ）が、追いかけっこをするようにして移動していくのです（図５－１－７）。

ここで扱うキューは、６バイト長のキューテーブルによって制御されます（図５－１－8）。このうち、プットポインタとゲットポインタは、キューバッファの先頭からのオフセット値を表わします。ですから、実際のメモリ上の番地は、これにキューバッファの先頭アドレスを加えたものになります。また、一度キューバッファから取り出したデータを１文字だけ戻すとき、直接キューバッファに戻すことはしません。そのようなデータを一時的に保管するための、バックキャラクタというものを用意してあります。キューの長さとしては、１、３、７、15、31、63、127、255バイトが指定できます。ただし、キューバッファの大きさはこれよりも１バイト余分に必要なので注意してください。

ところで、ふだんＰＣ－８８０１が扱うキューは２つあります。１つは、キー入力に使うものでキュー#０と呼ばれます。もう１つは、シリアル入力に使うもので、キュー#１といいます。キュー#０は、カセットとＲＳ－２３２Ｃとで共用されます。キー入力のキューバッファは、ＥＦＤ９Ｈから32バイト、カセット入力用のキューバッファはＦ０Ｄ３Ｈからの128バイトが割り当てられており、ＲＳ－２３２Ｃ用のキューにはファイルバッファ（５－１－２節参照）が割り当てられます。ＲＳ－２３２Ｃとカセットではキューバッファの領域が違うのですが、キューテーブルは同じ#１を使うため、同時に使用することはできません。

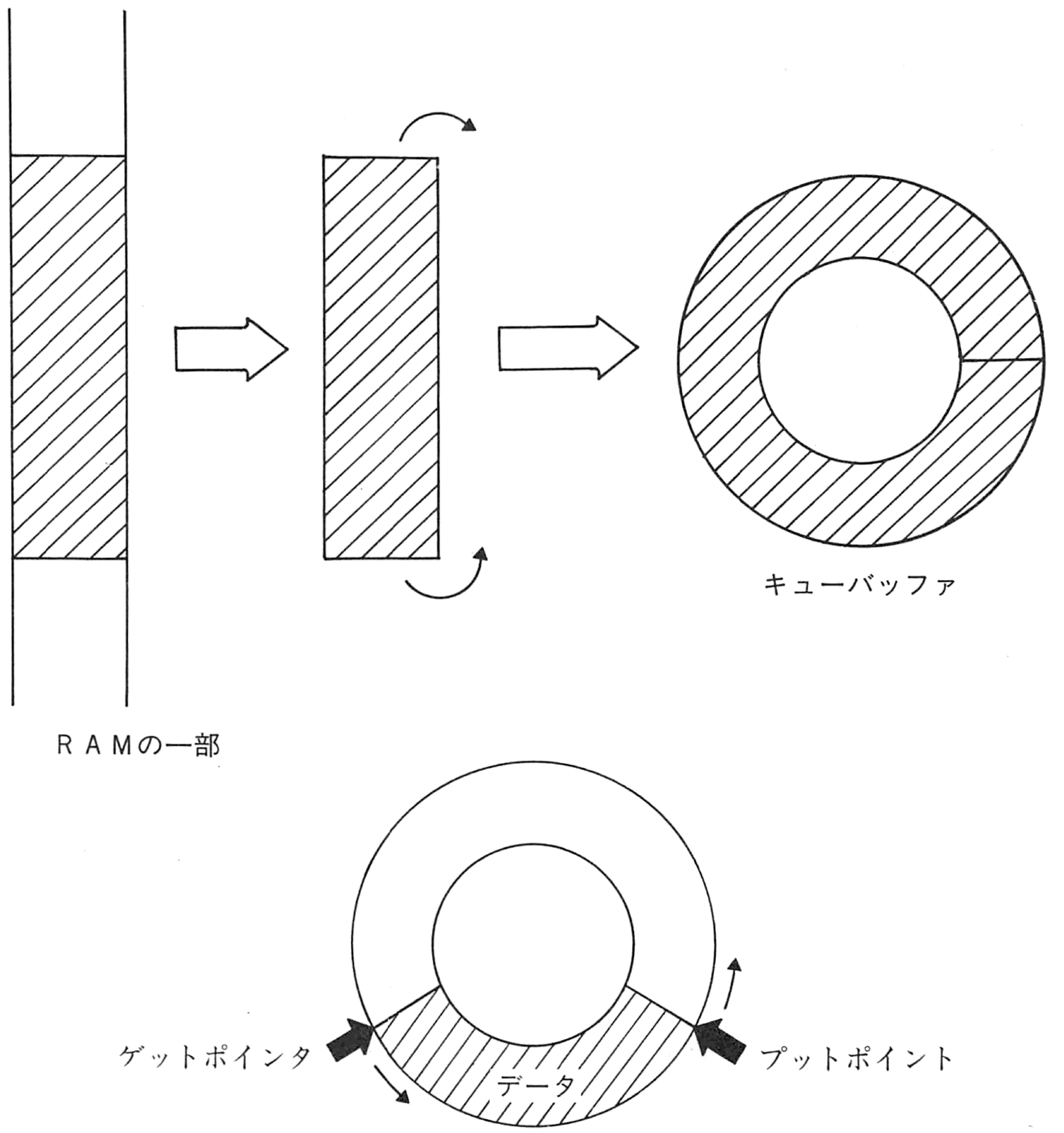


図５－１－７　キューバッファ

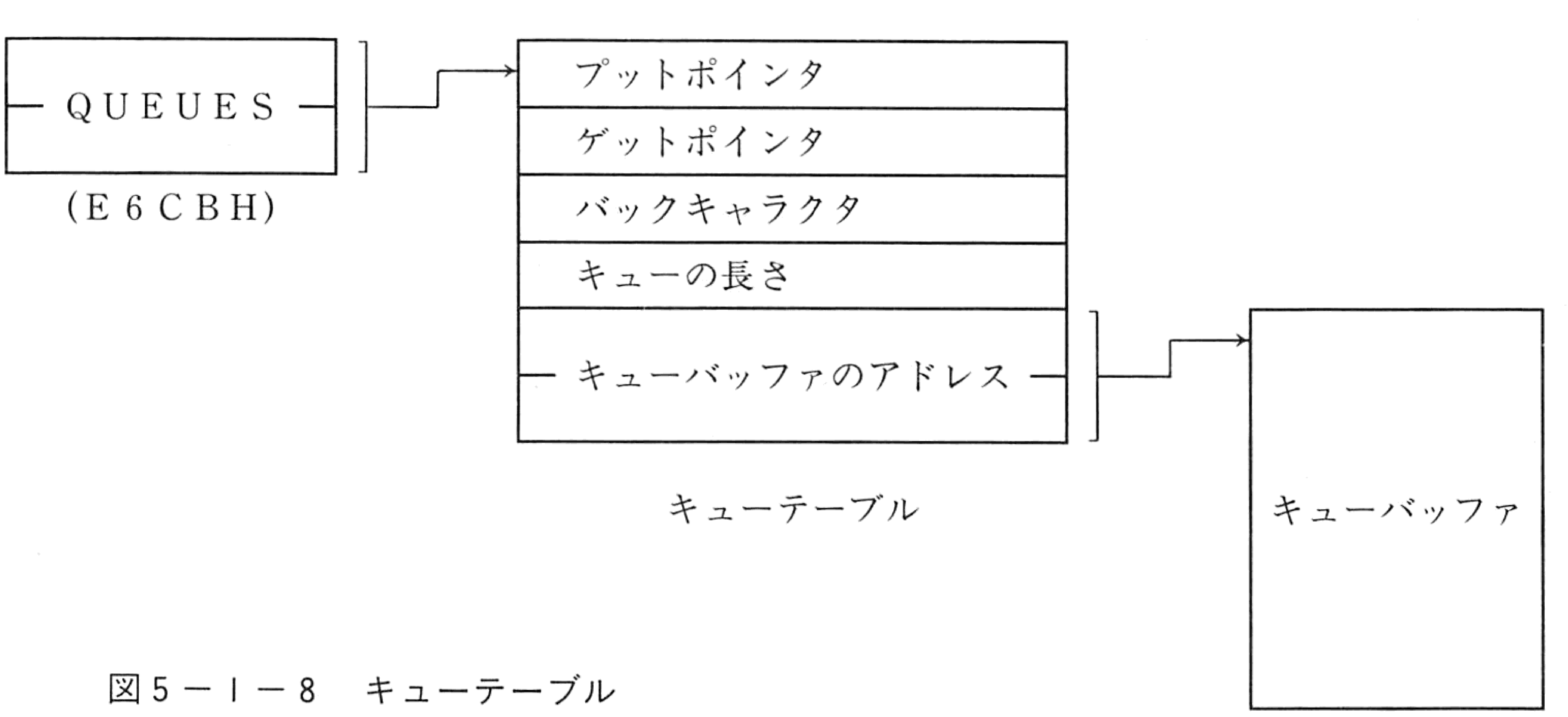


図５－１－８　キューテーブル

５－１－５　キュー処理ルーチン

キュー初期化ルーチン：ＩＮＩＴＱ

領　域	４６３ＤＨ
機　能	キューを初期化する
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｈ，Ｌ
入　力	Ａレジスタ　：キューの番号 Ｂレジスタ　：キューの長さ ＤＥレジスタ：キューバッファのアドレス
出　力	なし
ワーク	なし
ルーチン	４６８０Ｈ：キューテーブルのアドレスをＨＬレジスタに求める

キューテーブルのアドレスを求めた後、プットポインタ、ゲットポインタ、バックキャラクタをクリアして、キュー長とキューの先頭アドレスをセットします。

キュー書き込みルーチン：ＰＵＴＱ

領　域	４５ＤＤＨ
機　能	キューにデータを１文字書き込みます
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｈ，Ｌ
入　力	Ａレジスタ：キュー番号 Ｅレジスタ：データ
出　力	Ｚフラグ　：“１”のときキューがいっぱいで書き込めない
ワーク	なし
ルーチン	４６７６Ｈ：Ｂレジスタにプットポインタ、Ｃレジスタにゲットポインタ、Ａレジスタにバックキャラクタ、ＨＬレジスタにバックキャラクタのアドレスを入れる

キューがいっぱいでなければ、データを書き込んで、プットポインタを更新します。

キュー読み取りルーチン：ＧＥＴＱ

領　域	４５Ｆ８Ｈ
機　能	キューから１文字を得ます
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｈ，Ｌ
入　力	Ａレジスタ：キュー番号
出　力	Ａレジスタ：データ Ｚフラグ　：“１”のときキューが空
ワーク	なし
ルーチン	４６７６Ｈ：Ｂレジスタにプットポインタ、Ｃレジスタにゲットポインタ、Ａレジスタにバックキャラクタ、ＨＬレジスタにバックキャラクタのアドレスを入れる

バックキャラクタにデータがあれば、それをもってくるだけです（ただし、そのときバックキャラクタはクリアされる)。データがなければ、キューが空でないのを確認してから、ゲットポインタの次よりデータを取り込んでゲットポインタも更新します。

キュー内文字数計算ルーチン：ＮＵＭＱ

領　域	４６５５Ｈ
機　能	キュー内に入っている文字数を数える
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	Ａレジスタ：キュー番号
出　力	Ａレジスタ、ＨＬレジスタ：文字数
ワーク	なし
ルーチン	４６７６Ｈ：Ｂレジスタにプットポインタ、Ｃレジスタにゲットポインタ、Ａレジスタにバックキャラクタ、ＨＬレジスタにバックキャラクタのアドレスを入れる

［(プットポインタ)－(ゲットポインタ)］mod（キュー長)＋（バックキャラクタの数)
で計算されます。

キュー内空領域計算ルーチン：ＬＦＴＱ

領　域	４６６７Ｈ
機　能	キューにあと何文字書き込めるかを数える
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｈ，Ｌ
入　力	Ａレジスタ：キュー番号
出　力	Ａレジスタ、ＨＬレジスタ：キュー内の残り領域数
ワーク	なし
ルーチン	４６７６Ｈ：Ｂレジスタにプットポインタ、Ｃレジスタにゲットポインタ、Ａレジスタにバックキャラクタ、ＨＬレジスタにバックキャラクタのアドレスを入れる

［(ゲットポインタ)－((プットポインタ)＋１） mod（キュー長)］mod（キュー長）
により求めます。

キューバックキャラクタルーチン：ＢＣＫＱ

領　域	４６４ＥＨ
機　能	キューから得た文字を１つだけ戻す（連続して使うことはできない）
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｈ，Ｌ
入　力	Ａレジスタ：キュー番号 Ｅレジスタ：戻すキャラクタ
出　力	なし
ワーク	なし
ルーチン	４６８０Ｈ：キューテーブルのアドレスをＨＬレジスタに求める

キューテーブルのバックキャラクタに返すだけです。このとき、バックキャラクタが空かどうか確認していませんので、続けてこのルーチンを呼ばないように注意が必要です。

キュー書き込み取り消しルーチン：ＰＯＰＱ

領　域	４６１ＢＨ
機　能	キューに書き込んだものを無効とする
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｈ，Ｌ
入　力	Ａレジスタ：キュー番号
出　力	なし
ワーク	なし
ルーチン	４６１５Ｈ：Ｚフラグを０にする ４６７６Ｈ：Ｂレジスタにプットポインタ、Ｃレジスタにゲットポインタ、Ａレジスタにバックキャラクタ、ＨＬレジスタにバックキャラクタのアドレスを入れる

キューにデータがあれば、プットポインタを１つ前の状態に戻してキューからデータを取り戻します。もし、キューが空であれば、バックキャラクタから文字を取り出し、そこをクリアして戻ります。

5－2　キーボード処理ルーチン

キーボードキュー初期化ルーチン：KEYINI

領　域	35D9H
機　能	キーボード用のキュー#0を初期化する
破　壊	A，F，B，C，D，E，H，L
入　力	DI命令（Disable Interrupt）を実行しておく
出　力	なし
ワーク	EFD9H（KIQADR）：キュー#0バッファ
ルーチン	463DH（INITQ）　：キュー初期化ルーチン

バッファアドレスをEFD9H、キューの長さを31バイトとして、キー入力用キュー#0を初期化します。

キーボード1文字入力ルーチン：CHGET

領　域	3583H
機　能	キーボードから1文字入力する
破　壊	A，F
入　力	なし
出　力	Aレジスタ：キャラクタコード
ワーク	E6A5H（CLFLG）　：ターミナルモード用フラグ
	E6A7H（CURFG）　：カーソル表示フラグ
	E6BAH（F．COPY）：コピーモードフラグ
	E6CAH（INTFLG）：ctrl－O，ctrl－S，ctrl－C　検出用フラグ
ルーチン	35A8H：COPY処理ルーチン
	4290H（CURON）：カーソルオンルーチン
	45F8H（GETQ）　：キューから1文字入力するルーチン
	7780H：ターミナルモード時のバッファプリント処理

まず、INTFLGを調べます。この値が0となっていて(つまり、ctrl－O，ctrl－S，ctrl－Cが検出されていない)、しかも、CURFGが0でないならば、CURONを呼んでカーソルを表示します。次に、7780Hからのルーチンを呼びます。ここで、CLFLGの値によっ

て処理が別れます。ＣＬＦＬＧが「０」の場合には何もしません。「１」の場合にはプリンタにＣＲ・ＬＦを出力します。「２」の場合には、行バッファの内容をプリントします。いずれの場合においても、復帰するときにＣＬＦＬＧをクリアします。続いて、画面コピーの処理を行います。Ｆ．ＣＯＰＹが０でなく、コピーモードとなっている場合には、４thＲＯＭ内のルーチンを使ってハードコピーを実行します。以上の処理の後、ＩＮＴＦＬＧをクリアし、ＧＥＴＱを呼んでキューから文字を入力します。ただし、キューが空のときには、キューに文字が入力されるまで待ち続けます（図５－２－１）。

キースキャンルーチン：ＧＥＴＫＥＹ

領　域	３２４２Ｈ
機　能	キーボードマトリクスをスキャンしてキーの状態を調べる
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	なし
出　力	Ｃフラグ　：“１”のときキーが押されていない Ｃフラグ、Ｚフラグ：それぞれが“０”、“１”のとき以前と同じキーが押され続けている
ワーク	３０６９Ｈ：シフトキー変換テーブル ３３Ｆ６Ｈ：キー処理ジャンプテーブル Ｅ６Ａ２Ｈ（ＴＲＭＬＰＴ）：ターミナルモード時プリンタイネーブルフラグ Ｅ６Ａ５Ｈ（ＣＬ．ＦＬＧ）：ターミナルモード用フラグ Ｅ６Ｂ６Ｈ（Ｆ．ＬＴＲＬ）：コントロールコード表示フラグ Ｅ６ＢＡＨ（Ｆ．ＣＯＰＹ）：コピーモード Ｅ６ＣＥＨ（Ｆ．ＫＹＳＴ）：ファンクションキーフラグ Ｅ６ＣＦＨ（ＫＥＹＰＲＴ）：キー入力リピートカウンタ Ｅ６Ｄ０Ｈ（ＫＥＹＢＩＴ）：第一キーステータステーブル（12バイト） Ｅ６ＤＣＨ（ＫＥＹＶＬＤ）：第二キーステータステーブル（12バイト） Ｅ６Ｆ２Ｈ（ＳＴＲＴＡＢ）：ファンクションキーの内容（15×16バイト） ＥＦ７２Ｈ（ＴＲＭＨＬＦ）：ターミナルモード用全二重／半二重フラグ ＥＦ７５Ｈ（ＬＩＴＦＬＧ）：ターミナルモード用リテラルフラグ ＥＦＦ９Ｈ（Ｆ．ＫＳＣＮ）：キースキャンフラグ。この値がそれぞれ、 ２　：新しいキーを押した、 ３　：キーを押していない、 ＦＦＨ：キーを押したまま、であることを表す ＥＦＦＡＨ（ＮＫＥＹＢＴ）：入力キー位置指示フラグ

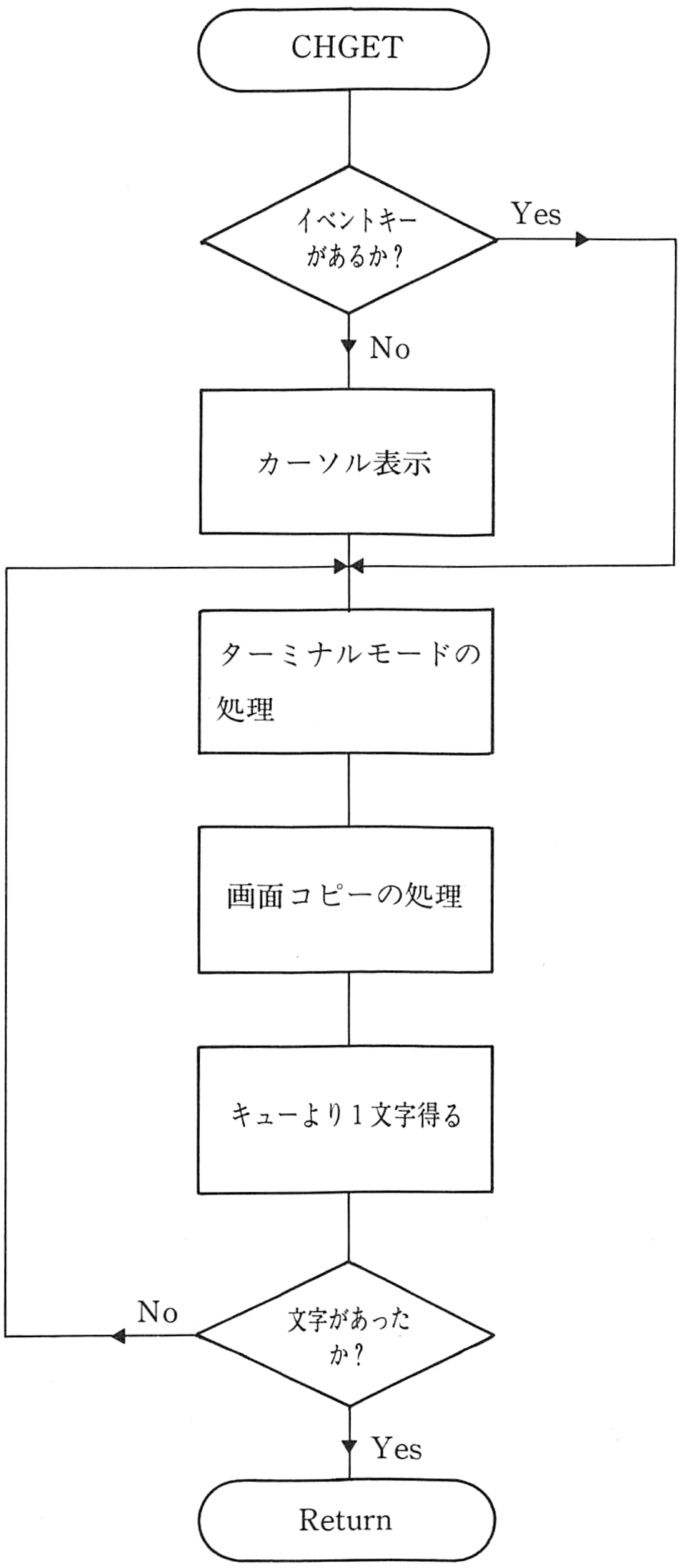


図5－2－1　CHGETルーチン

EFFBH(NKEYAD)：第二キーステータステーブル中の位置を示すフラグ
EFFDH (KEYCOD)：キーコード保存フラグ
EFFEH (ASCKEY)：入力したキーのASCIIコード
EFFFH (SHFBIT)：シフトデータ
F000H (OLDSHF)：前回のシフトデータ
F001H (CAPLOK)：CAPSキーフラグ
F002H (SHFVAL)：シフトの種類
F00CH (PFKYNM)：以前に押されたファンクションキーの番号

ルーチン　325CH：コード変換
32F1H：ポートのスキャン
335CH：キーコードの変換
3F80H：ファンクションキーの表示
ECDFH：フック (RET)

PC－8801のキーボードは、図5－2－2のように、I／Oのアドレスとデータバスの交点にキーが配置してあるだけで、コード変換のためのハードウェア上の用意は為されておりません。そのため、ソフトウエアの側でキーの状態を読み取り、JISコードに変換する必要があります。この処理を行うのが本ルーチンです。なお、アドレスバスとキーマトリクスの間のデコーダICの出力は負論理となっています。したがって、データバスに現れる値はキーを押した時が"0"、キーを押していないときが"1"となりますので注意が必要です。

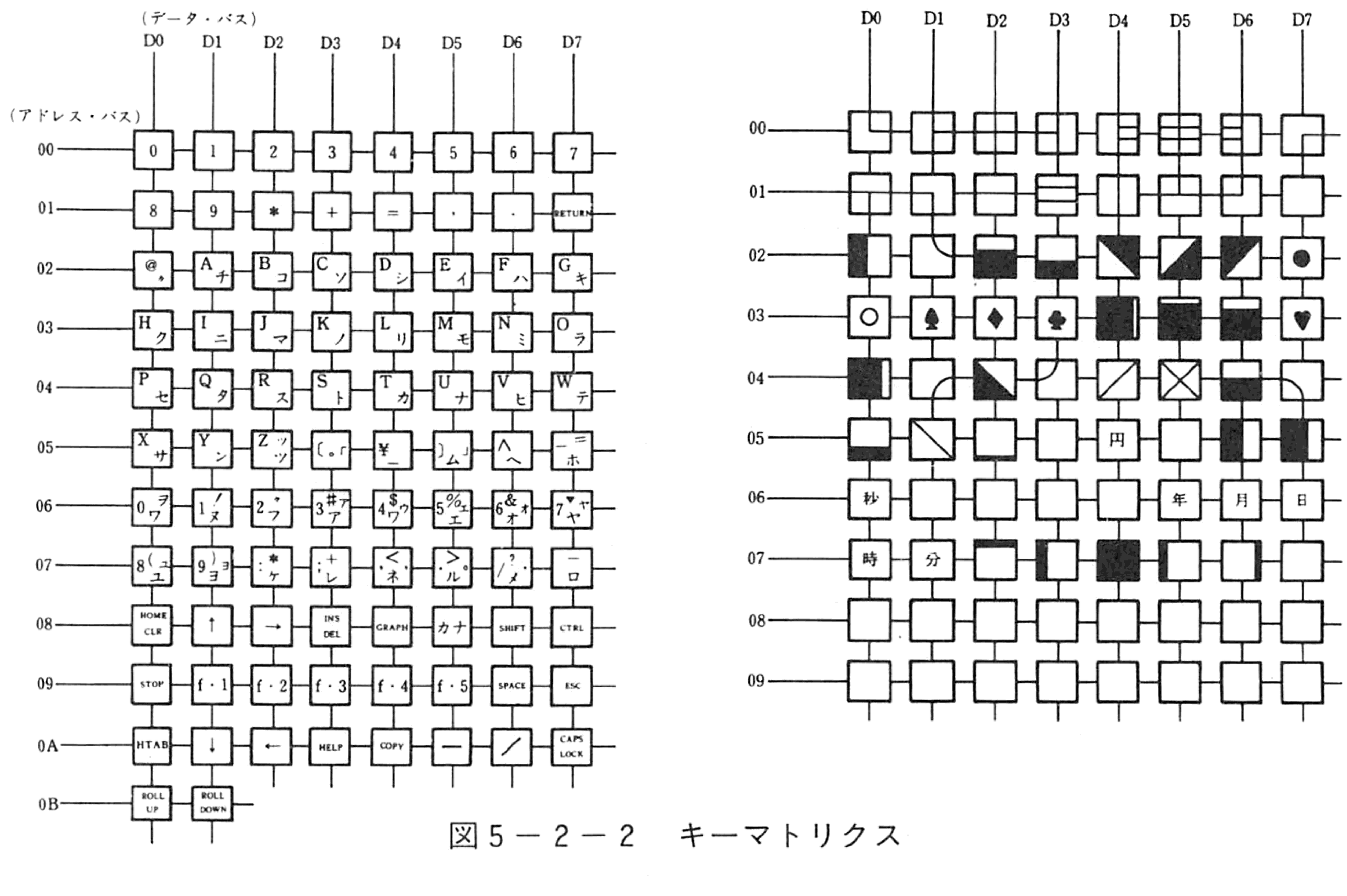


図5－2－2　キーマトリクス

まず、初めにすべてのキーの状態を読み取りますが、これについてはあとで詳しく説明します。次に、各シフトキー（ＣＴＲＬ，ＳＨＩＦＴ，カナ，ＧＲＰＨの各キー）の状態変化を調べます。各シフトキーの押され方が前回と異なるときは、その組み合わせが適当であるかどうか調べます。これらのキーは同時に押して使うことができますが、一般的なのは、「ＳＨＩＦＴ＋カナ」の場合です。その他、「ＣＴＲＬ」がからんだ組み合わせでは、「ＣＴＲＬ」が最優先で扱われます。また、「ＣＴＲＬ」がなく、「ＧＲＰＨ」と「カナ」、あるいは「ＧＲＰＨ」と「ＳＨＩＦＴ」が同時に押された場合には、「ＧＲＰＨ」が優先されます。それから、ファンクションキーを画面下部に表示します。ただし、「ＳＨＩＦＴ」キーの状態により、「ＰＦ１」～「ＰＦ５」か「ＰＦ６」～「ＰＦ10」のどちらかが選ばれます。

キーの状態に変化がなかったとき、すなわち、キーが押されなかったり、以前と同じキーが押されたままのときには、このルーチンは終わりとなります。その際、出力条件に適合するように、ＣＰＵのフラグがセットまたはリセットされます。キーの状態に変化が認められた場合には、第二キーステータステーブルを更新します（これについてはキースキャンのところで詳しく説明します)。その後、ＪＩＳコードへの変換をします(これも後述)。最後に、「CAPS−LOCK」の処理をします。「ＣＡＰＳ」キーがロックされていて、アルファベットの大文字が押された場合、小文字に変換されます。小文字が押された場合には大文字に直されます。以上が、このルーチンの大まかな流れです。

それではキースキャンについて述べます。フローチャートを図５－２－４に示します。このルーチンでは、下記の処理をキーが接続されているすべてのポート（００Ｈ～０ＢＨ）に対して行います。

最初に、各ポートのキーの状態と第一キーステータステーブルとの比較を行います。このルーチンではキーの状態を管理するためにテーブルを２つ使用しますが、この第一キーステータステーブルには、以前に読んだ各ポートからのデータを反転した値が書き込まれています。そこでこれらの値と比較した後、これを更新します。変化があった場合には、即座にこのルーチンから復帰します。

次に行うのは、第二キーステータステーブルの更新です。このテーブルはキーが押され続けている場所(ビット)が“１”に、それ以外は“０”になるように処理されています。なお、図５－２－３における更新では、新しく押されたキーの位置を“１”にしています。

それから、特殊キーをフラグにセットします(図５－２－５参照)。特殊キー以外の押下がなければ、このルーチンはここで終わりです。それらの押下があり、第二キーステータステーブルとの相違もあり、かつ、Ｆ．ＫＳＣＮが「２」である（今調べているポートよりアドレスの小さいポートでのキーの変更があった）場合には、キーコードを計算してＫＥＹＣＯＤに格納します。ここで、キーコードというのは押されたキーのポート上の位置を示す情報で、図５－２－６のような構成となっています。例えば、“A”のキーであれば、そのキーコードは０００

１０００１Ｂ（＝１１Ｈ）となります。これは、次に述べるコードの変換において使用されます。

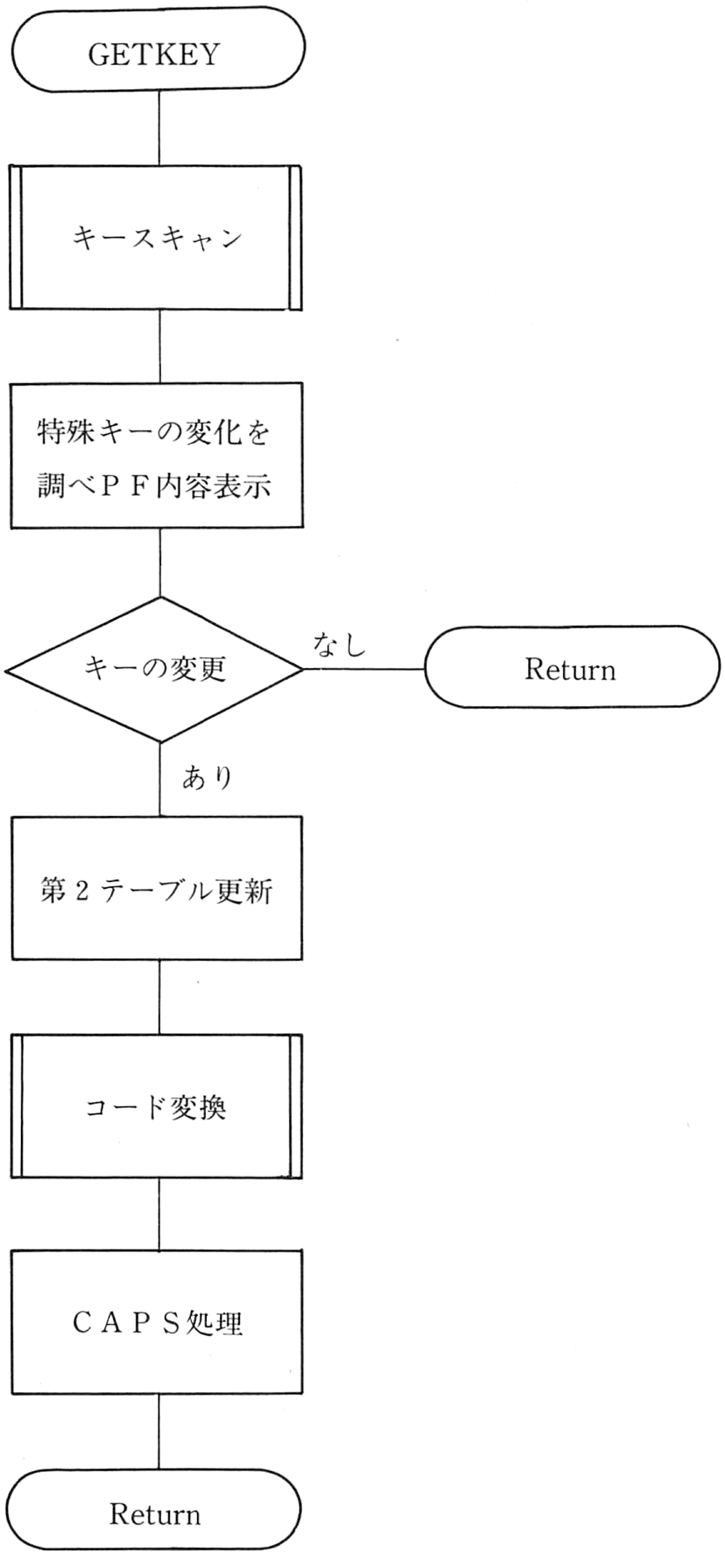


図５−２−３　ＧＥＴＫＥＹルーチンのフローチャート

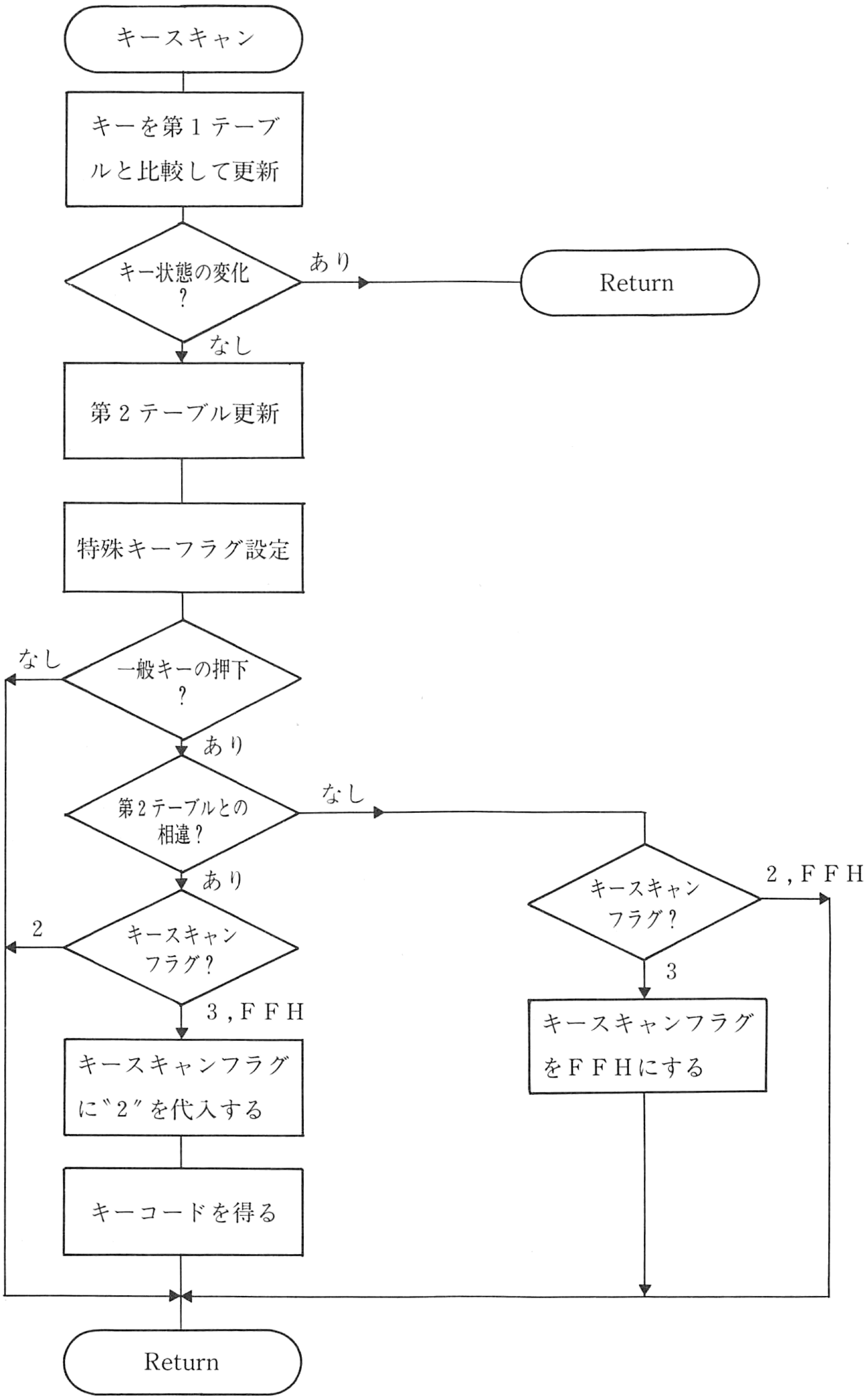
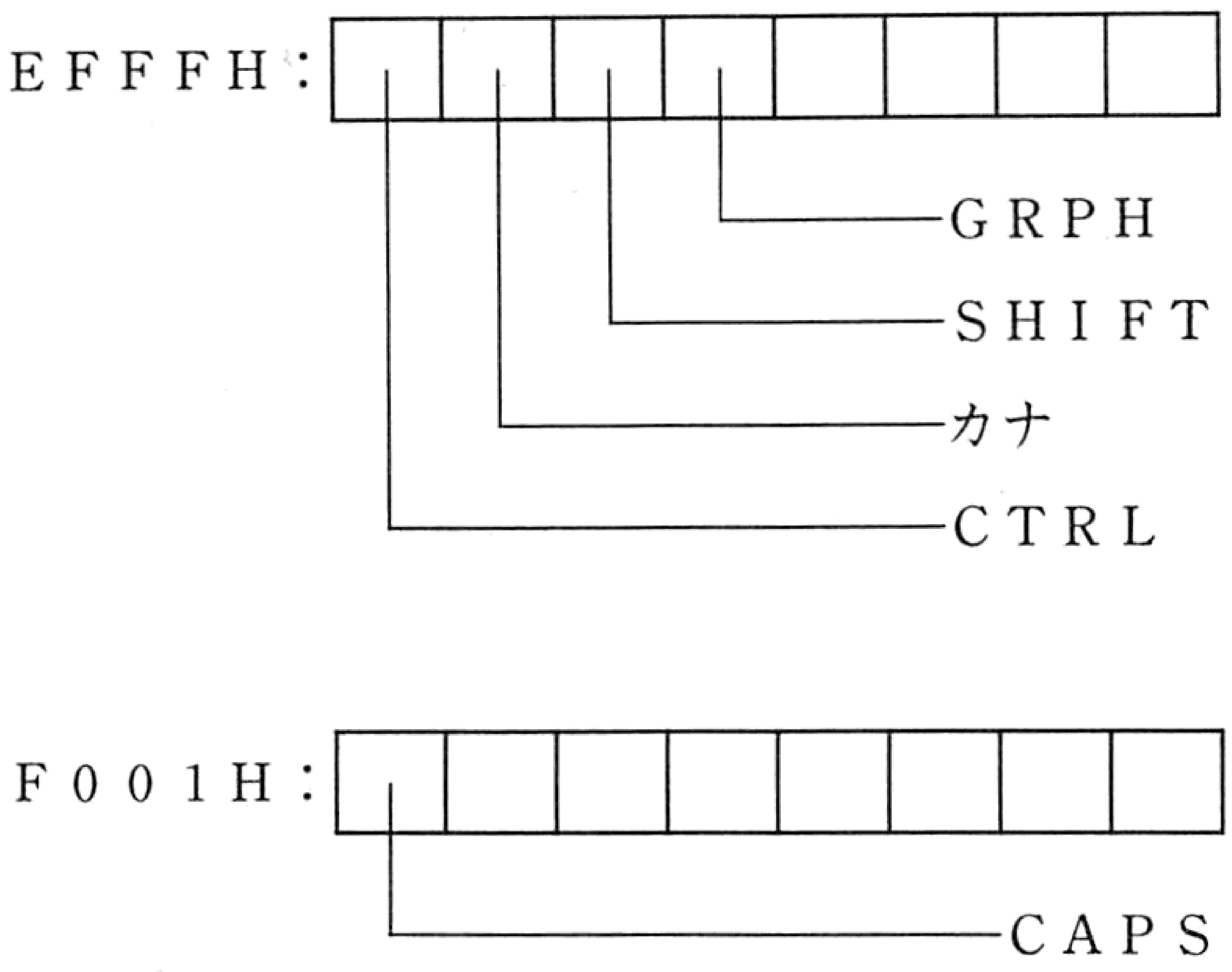


図５－２－４　キースキャンルーチン



それぞれのビットが
1：対応する特殊キーが押されている
0：対応する特殊キーが押されていない

図５－２－５　特殊キーフラグ

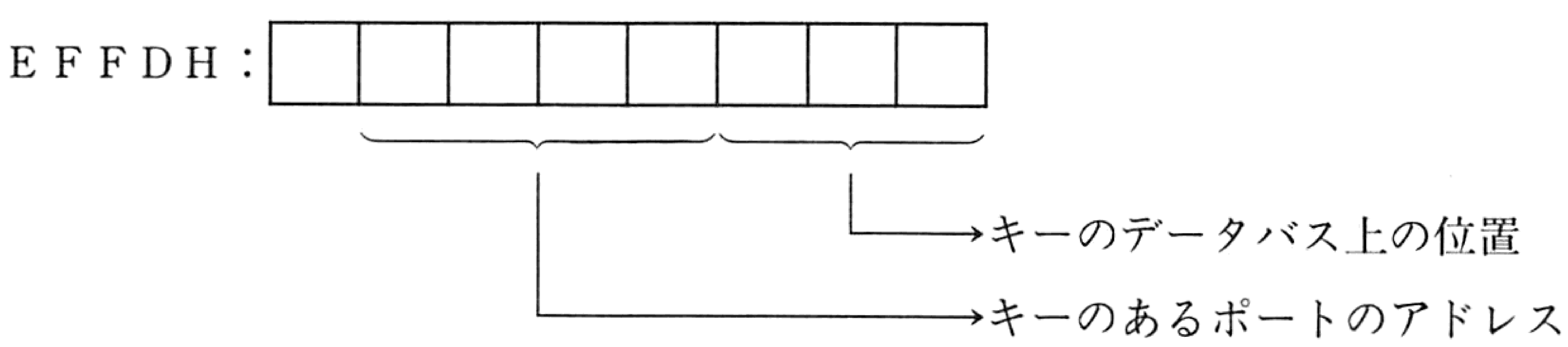


図５－２－６　キーコード

コード変換ルーチンでの処理は図５－２－７にあるように表から求めるものとそれ以外のものとに分類することができます。「ＨＥＬＰ」,「ＥＳＣ」,「ＳＰＡＣＥ」,「ＳＴＯＰ」の場合には、そのＪＩＳコードを直接Ａレジスタに代入するだけで終わりです。「ＣＯＰＹ」の場合には、Ｆ．ＣＯＰＹにコピーコードを表す値を入れます（ＦＦＨのときノーマル、ＦＤＨのときテキスト、ＦＡＨのときグラフィック）。これは、シフトナンバー（図５－２－８）を反転したものですので、「ＣＯＰＹ」キーを押してハードコピーをとる場合、「ＣＴＲＬ」＋「ＣＯＰＹ」（テキスト画面のみ)、「ＧＲＰＨ」＋「ＣＯＰＹ」（グラフィック画面のみ）といった使い方もできます。その後、現在ターミナルモードであれば、キュー＃０が初期化され、ctrl－Ｍ（一時停止）が２つ送られます。

```
キーコード≧５０Ｈ                            （ＪＩＳコード）
        ┌≧５ＡＨ------エラー
        │＝５３Ｈ------ＨＥＬＰ              ０１Ｈ
        │＝５４Ｈ------ＣＯＰＹ              なし
        ┤＝５９Ｈ------ＲＯＬＬ　ＤＯＷＮ ┐
        │＝５８Ｈ------ＲＯＬＬ　ＵＰ      ┘表検索によって求める
        └それ以外------表検索によって求める
キーコード＜５０Ｈ                            （ＪＩＳコード）
        ┌＝４ＦＨ------ＥＳＣ                １ＢＨ
        │＝４ＥＨ----ＳＰＡＣＥ              ２０Ｈ
        ┤＝４８Ｈ------ＳＴＯＰ              ０３Ｈ
        └それ以外------表検索によって求める
```

図５－２－７　キーコードの変換

シフトナンバー	対応するキー
0	なし
1	ＳＨＩＦＴ
2	ＣＴＲＬ
3	カナ
4	カナ ＋ ＳＨＩＦＴ
5	ＧＲＰＨ

図５－２－８　シフトコード

「ROLL UP」と「ROLL DOWN」の場合、最終的には表からコードを得ることになりますが、途中でＯＮ割り込みの処理を行うルーチンを経由しております。これらのキーのためにもＯＮ割り込み用のテーブルが確保されているのですが、ＢＡＳＩＣでは使用しておりません。

上記以外のキーのＪＩＳコードは、表から求めるわけですが、その前に類似したキーをまとめて処理するための分岐があります。この処理のためのテーブル（３３Ｆ６Ｈ～３４９７Ｈ）を図５－２－９に示します。まず、キーコードを左端の分類コード（０Ａ，１０，……）と比較していき、キーコードの方が表の数字よりも小さければ、その列を選択します。次にＳＦＨＶＡＬに保存されているシフトナンバーから、求めるデータを見つけます。例えば、シフトナンバーが２ＡＨで「ＳＨＩＦＴ」が押されている場合、得られるデータは０９３５４０Ｈです。このとき、前の２バイトがＨＬレジスタに、最後の１バイトがＢレジスタに入ります。ま

た、Ｃレジスタには、１つ手前の分類コードの値（この場合は１０Ｈ）が入り、Ａレジスタにはキーコードが入ります。そして、ＨＬレジスタで示されるアドレスに飛んで（上の例では３５０９Ｈ)、それぞれの処理を行います。以下にそれらを示します。

３４Ｂ２Ｈ：エラー

３４Ｂ４Ｈ：ＡレジスタとＣレジスタが等しいとき、求めるコードは３０Ｈ等しくないとき、求めるコードは次式で与えられます：
(Ａレジスタ)－(Ｃレジスタ)＋(Ｂレジスタ)

３４ＢＡＨ：求めるコードは、(Ａレジスタ)－(Ｃレジスタ)＋(Ｂレジスタ)

３４ＢＤＨ：３４ＢＥＨ＋(Ｂレジスタ)で示されるアドレスから2バイトの値を取り出し、その値にＡレジスタの値を加えた値で示されるアドレスにあるデータが求めるコードとなります。

３４Ｄ２Ｈ：これはファンクションキーの処理をするルーチンです。
最初に、３４ＢＡＨのルーチンを使ってファンクションキーの番号（ただし０～９）を求め、ＰＦＫＹＮＭに保存しておきます。また、Ｅ６ＣＥＨにはＦＦＨを書き込み、ファンクションキーが押下を示しておきます。ＶＲＴＣ割り込みでは、これを参照して、ファンクションキーが押されたという情報を得ます。ファンクションキー内容のキーバッファ（キュー#０）への書き込み、ファンクションキー割り込みの処理の実行などは、ＶＲＴＣ割り込みルーチンの担当となります（6章参照)。

３５０１Ｈ：求めるコードは、(Ａレジスタ)－(Ｃレジスタ)＋(Ｂレジスタ)で与えられますが、これが６０Ｈとなった場合に限り、４０Ｈを求めるコードとします。

３５０９Ｈ：求めるコードは、(Ａレジスタ)－(Ｃレジスタ)＋(Ｂレジスタ)となりますが、これが４０Ｈとなった場合にのみ、７ＥＨをコードとします。３５０１Ｈと３５０９Ｈは、アルファベットの各キーＡ～Ｚを処理するものです。例外はそれぞれ「@」および「～」の場合です。この「～」(SHIFT＋「@」)はキートップに表示されていない“隠しキー”です。

３５１１Ｈ：「@」およびＡ～Ｚのキーが「カナ＋ＳＨＩＦＴ」の状態で押された場合の処理です。ここで問題となるのは、「ィ」と「ッ」の場合で、キーコードがそれらのキーを表す１５Ｈ，２ＡＨであれば、それぞれ、Ａ８Ｈ，ＡＦＨを求めるコードとします。それ以外のキーの場合には、シフトキーのあるなしにかかわらず、同じコードとなります。

３５１ＦＨ：キーコードが２ＦＨのとき、求めるコードは２ＤＨとなります。
それ以外の場合には、３４ＢＡＨの処理を行います。

キーの種類	分類コード	ノーマル	SHIFT	CTRL	カナ	カナ・SHIFT	GRPH	Cレジスタの値
[0][1]----------[9] 〔テンキー数字〕	0A	BD34 (30)	BA34 (30)	BA34 (30)	BA34 (30)	BA34 (30)	BA34 (00)	00
[*][+][=][,][.][⏎]	10	BD34 (04)	BD34 (04)	BD34 (04)	BD34 (04)	BD34 (04)	BD34 (02)	0A
[@][A][B]------[Z]	2B	0135 (60)	0935 (40)	BA34 (00)	BD34 (06)	1135 (06)	BD34 (08)	10
[[][¥][]][^][_]	30	1F35 (5B)	BD34 (0A)	B234 (00)	BD34 (0C)	2635 (0C)	BD34 (0E)	2B
[0][1]----------[9] 〔フルキー数字〕	3A	BA34 (16)	B434 (20)	B234 (00)	BD34 (10)	BD34 (12)	BD34 (14)	30
[:][;][,][.][/][−]	40	BD34 (16)	BD34 (18)	B234 (00)	BD34 (1A)	BD34 (1C)	BD34 (1E)	3A
[HOME][↑][→][INS][GRPH] [カナ][SHIFT][CTRL][STOP]	49	BD34 (20)	BD34 (22)	B234 (00)	BD34 (20)	BD34 (22)	B234 (00)	40
[PF1]-------[PF5]	4E	D234 (00)	3435 (05)	B234 (00)	B234 (00)	3435 (05)	B234 (00)	49

図5－2－9　キーコード分類テーブル

※各欄の意味　HL代入値
(B代入値)

３５２６Ｈ：キーコードが２ＢＨ、２ＤＨの場合を特に取りあげ、それぞれに"「"、"」"のコードである、Ａ２Ｈ，Ａ３Ｈを与えます。それ以外では３４ＢＤＨの処理へと移ります。

３５３４Ｈ：ファンクションキーがシフトキーとともに押された場合の処理です。もし、ＢＡＳＩＣモードであれば、３４Ｄ２Ｈの処理へ移ります。ターミナルモードの場合は、各キーの番号により以下の処理をします。ＰＦ６の場合は、コントロールコードを実行するか、実行せずに文字として表示するかの切り換えをします(トグル動作)。そのためにＬＩＴＳＬＧとＮ．ＬＴＲＬを反転します。ＰＦ７の場合には、全二重／半二重の切り換えをします。これもトグル動作で、ＴＲＭＨＬＦを反転します。ＰＦ８の場合には、画面の出力をプリンタにも出力するかどうかの切り換えをします。ここでは、ＴＲＭＬＰＴを反転します。ＰＦ９はバッファの内容をプリンタへ出力するもので、ＣＬ．ＦＬＧに「２」を書き込みます。ＰＦ１０はプリンタを１行フィードするもので、ＣＬ．ＦＬＧに「１」を書き込みます。ただし、ＴＲＭＬＰＴが0でない場合、すなわち、ＰＦ８などによりプリンタの使用が許可されていない場合には何もしません。

ブレークチェックルーチン：ＢＲＫＣＫ

領　域	３５Ｃ２Ｈ
機　能	ＳＴＯＰキーが押されていないか調べる
破　壊	Ａ，Ｆ
入　力	なし
出　力	Ｃフラグ：１のときＳＴＯＰキーを検出
ワーク	Ｅ６ＣＡＨ（ＩＮＴＦＬＧ）：イベントキーフラグ
ルーチン	なし

ＩＮＴＦＬＧが「３」または「４」であるとき、キャリーフラグをセットします。ＳＴＯＰキーが押されると、このフラグがセットされますので、このルーチンをうまく使えば、ＳＴＯＰキーが押されたかどうか調べることができます。

5－3　プリンタ処理ルーチン

プリンタ出力ルーチン：ＬＰＴＣＨＲ

領　域	３ＥＤ４Ｈ
機　能	プリンタに１文字出力します
破　壊	Ａ，Ｆ
入　力	Ａレジスタ：プリントするキャラクタコード
出　力	なし
ワーク	Ｅ６Ｃ１Ｈ（Ｓ．ＳＹＣ１）：ポート４０Ｈに出力したデータ
ルーチン	３５Ｃ２Ｈ（ＢＲＫＣＫ）　：ブレークチェック

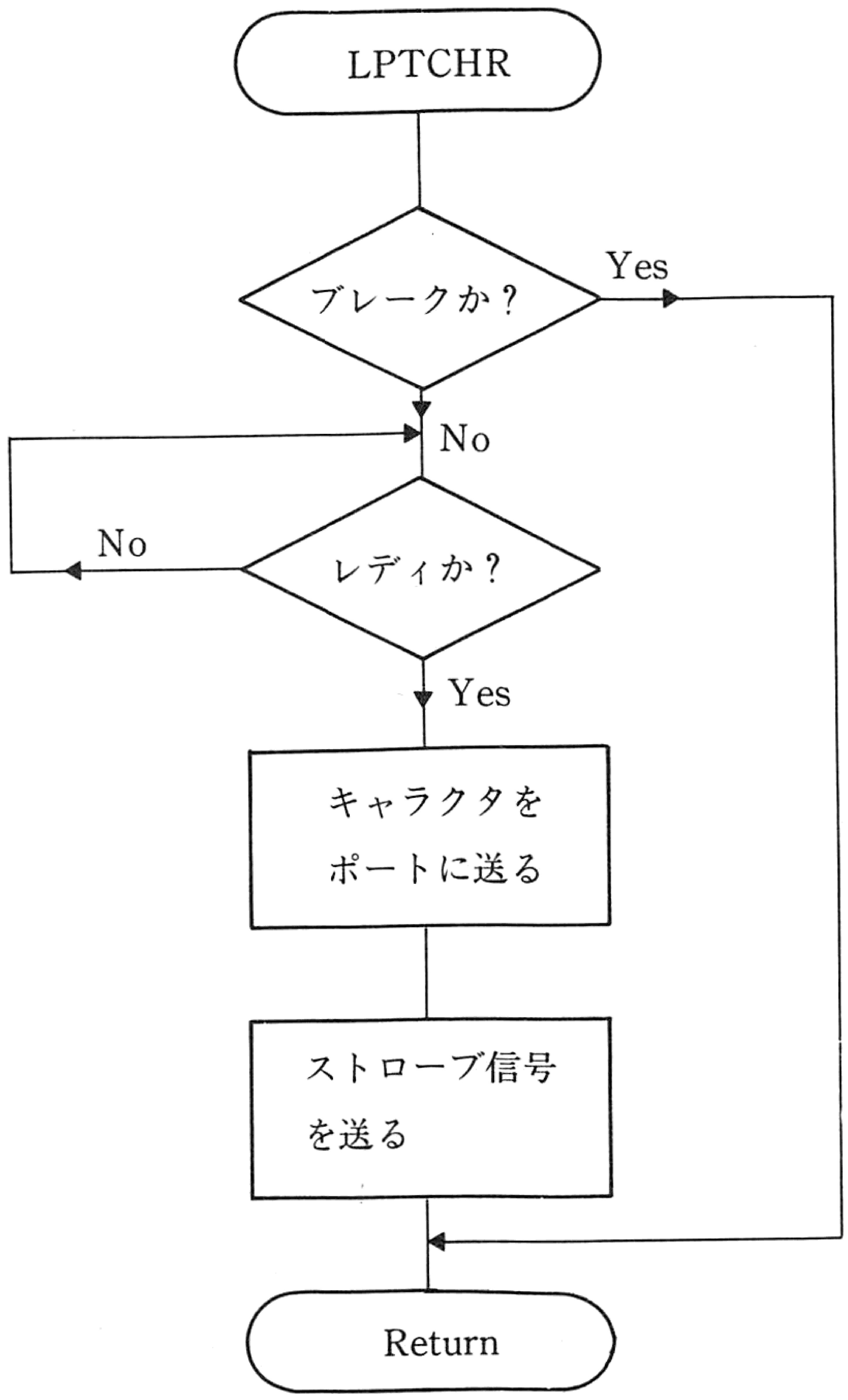


図５－３－１　ＬＰＴＣＨＲルーチン

ポート４０Ｈビット０のプリンタＢＵＳＹを調べ、Ready(ＢＵＳＹ＝"０")になるまで待ちます。Readyになったら、ポート１０Ｈにデータを出力します。そして、その直後にストローブ信号を送ります。ストローブ信号はポート４０Ｈビット０に送りますが、これは図５－３－２のようなパルスでなくてはなりません。そのため、一度"０"にした後、すぐに"１"に戻します。なお、このルーチンではブレークチェックを行っていますので、Readyを待つ間にＳＴＯＰキーが押されると、キャリーフラグだけをセットしてリターンします。

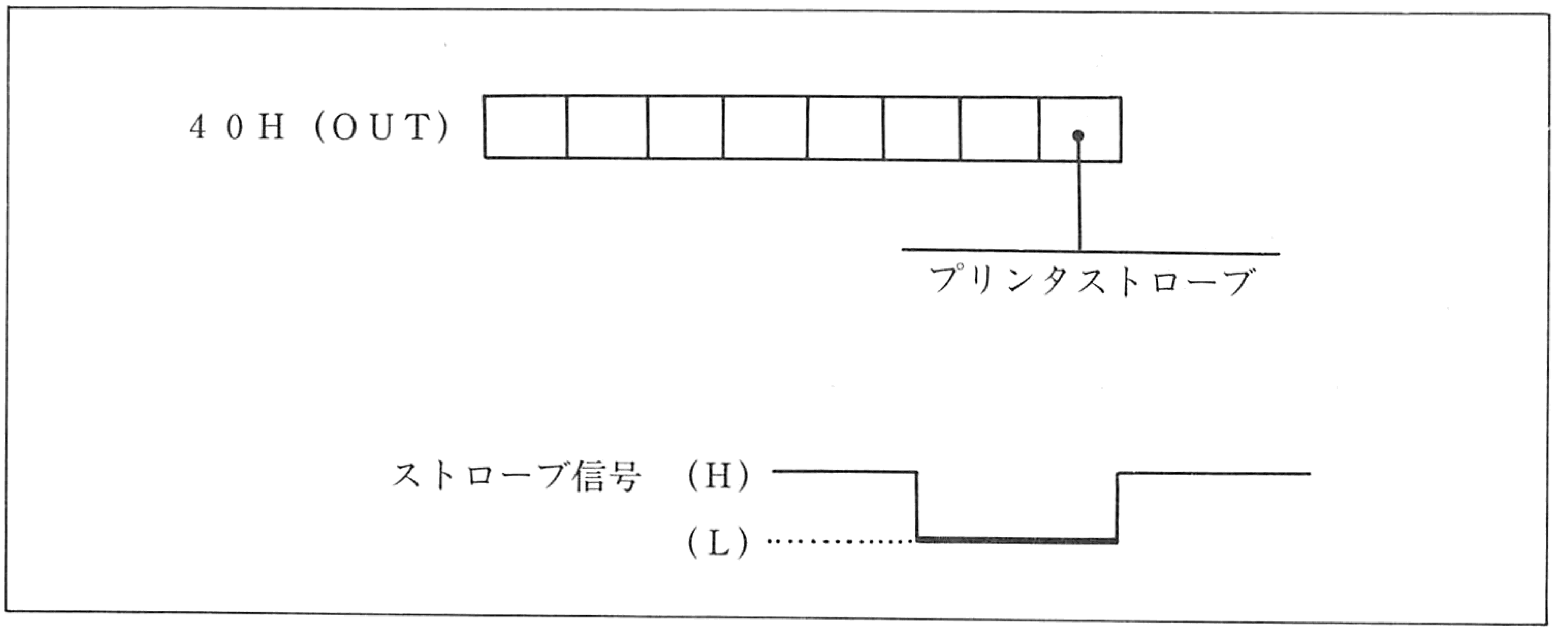


図５－３－２　プリンタストローブ信号

5－4　スクリーン

5－4－1　スクリーンの構成

テキスト画面用ＶＲＡＭの構成は図５－４－１のようになっています。ただし、40桁モードのとき使用される表示データエリアは、偶数アドレスのものだけです（つまり、メモリが１つおきに使われることになります)。この表示データエリアというのは、表示するキャラクタそのものを書き込むところであり、それらを表示する際の属性はアトリビュートエリア内のデータによって決められます。アトリビュートデータは、アドレス部と属性部１バイトずつが１組となって使われます。直前のアトリビュートから数えて、アドレス部が示す数のところまでの文字が、属性部が示すアトリビュートコードによって支配されます。(図５－４－２参照、アトリビュートコードについては図５－４－５参照)。なお、40桁モードでアドレス部を計算する際には、画面に表示されない奇数番地の分も数えなくてはいけないので注意が必要です。

ヌルキャラクタバッファとヌルアトリビュートバッファには、画面の１行を空白にするときのコードが入っています。これらの領域はＶＲＡＭとはいえませんが、画面上のある行をクリアするときに使ったりするので、ここに載せておきました。

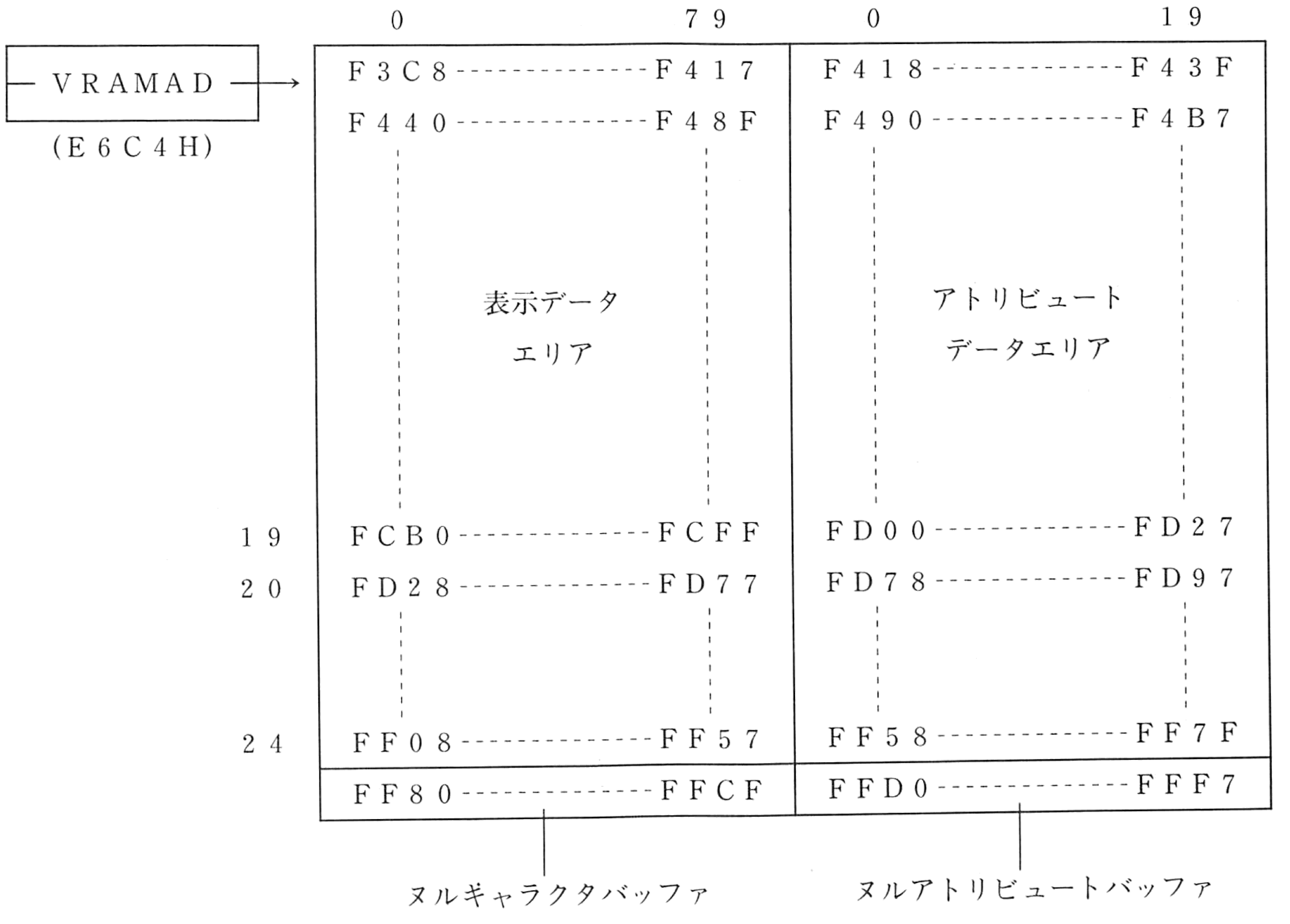


図5－4－1　テキストVRAMの構成

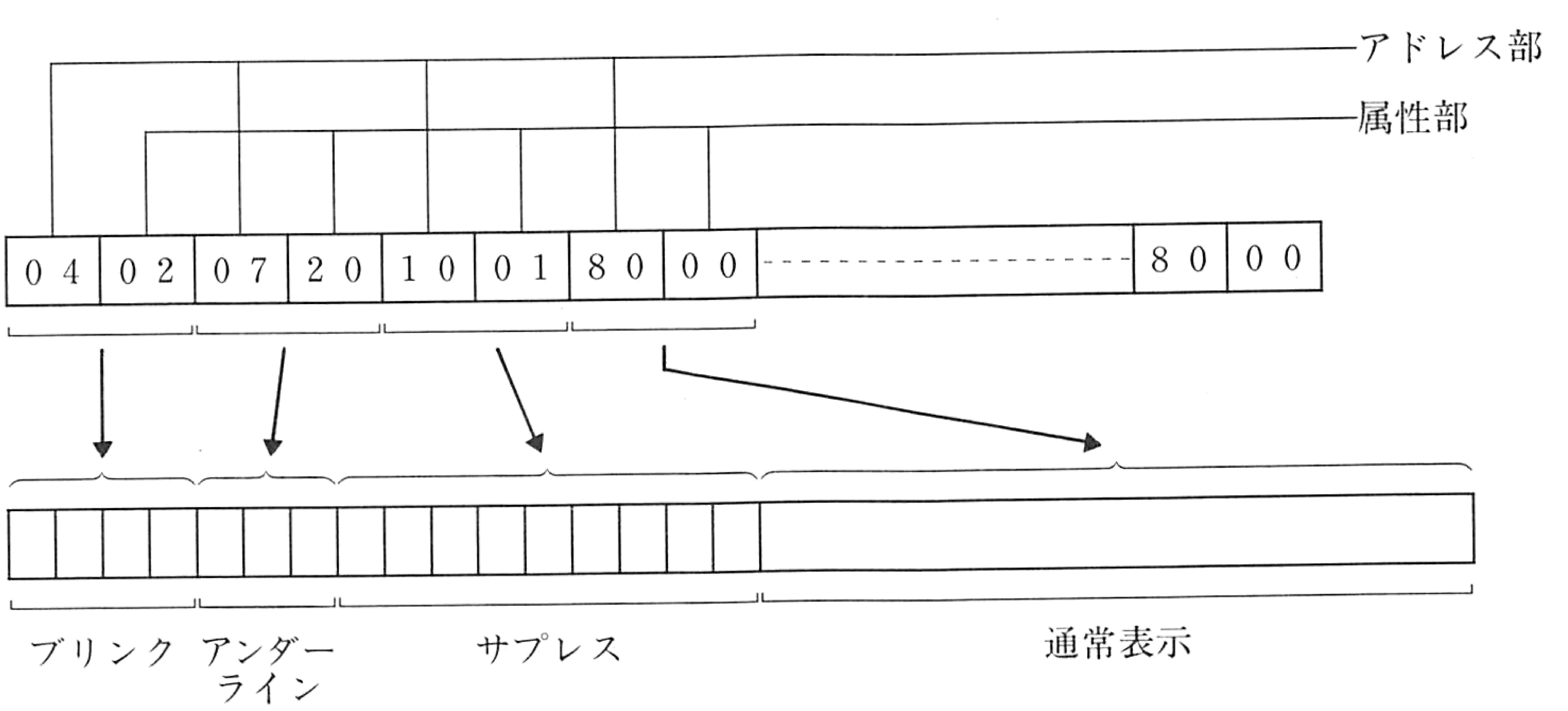


図5－4－2　アトリビュートエリア

5−4−2　スクリーン処理ルーチン

CRTモード変更ルーチン：CHGMD1

領　域	6F6BH
機　能	CRTのモードを変更する
破　壊	A，F，B，C，D，E
入　力	E6B2H（SCRLL1）　：スクロール開始行 E6B3H（SCRLL1＋1）：スクロール終了行 E6B9H（CMOD）　：B／Wモードのとき0、カラーモードのときFFH Bレジスタ：1行あたりの文字数 Cレジスタ：1画面あたりの行数
出　力	なし
ワーク	E64FH（LINLEN）：画面の桁数 E650H（CLMLST）：最終プリント位置 E6B0H（SCRCC）　：実際のスクロール開始行 E6B1H（LINEND）：実際のスクロール終了行 E6B2H（SCRLL1）：スクロール開始行 E6B3H（SCRLL2）：スクロール終了行 E6B8H（CNSDFG）：ファンクションキー表示フラグ E6B9H（CMODE）　：カラーモード E6C1H（S．SYC1）：ポート40に出力したデータの控え E6C0H（S．SYC0）：ポート30Hに出力したデータの控え E6C2H（S．CRTC）：ポート31Hに出力したデータの控え E6C4H（VRAMAD）：VRAMスタートアドレス EF88H（LINCNT）：画面行数 EF89H（LINWDT）：画面桁数
ルーチン	1880H：CLMLSTの設定 6FB4H：スクロール範囲の調整 6FD1H：CRTC、DMACへのパラメータの設定

最初に、6FB4Hからのルーチンを使ってカーソルのY座標およびスクロールの上限、下限を調整します。このルーチンに入る際のCレジスタの値から、CNSDFG（ファンクショ

ンキーを表示するなら１、表示しないなら０）を引いた値と、現在のカーソルのY座標とを比較します。もしこのY座標のほうが範囲よりも大きいのであれば、Y座標をその範囲内で最大となるよう変更します。次に、設定したい表示行数とＬＩＮＥＮＤ，ＳＣＲＬＬを比較し、現在の各行の方が大きい場合はそれぞれを表示行数の値に変えます。

これが済んだらＬＩＮＣＮＴ，ＬＩＮWＤＴの設定をします。また同時に、ＣＬMＬＳＴの更新も行います。ＰＲＩＮＴ文を使って画面に文字を表示する際、引数を“，”で区切ってたくさん並べると、１行内には最大２つ（40桁モード時）もしくは５つ（80桁モード時）置くことができますが、これらの内で一番最後の要素のその行における先頭からのオフセットをＬＩＮWＤＴが表します（図５－４－４参照）。実際には、40桁モードの場合に０ＥＨ、80桁モードでは３８Ｈとなります。

それから、ＣＲＴＣとDMAＣの設定を６ＦＤ１Ｈからのルーチンで行います。その内容は以下のとおりです。

（１）　高解像度ＣＲＴにおける行数（20／25）設定（ポート３１Ｈビット５）

（２）　80桁モード、カラーモード設定

（３）　ＣＲＴＣのＩＦ同期コントロールビットオン

次に４３３ＦＨのルーチンでＣＲＴＣからの垂直帰線信号がディスプレイサイクルから垂直帰線サイクルに切り換わったのを確認したうえで、以下の処理へと移ります。

（４）　ＤMＡＣをオートロードモードにセット（ＣＲＴが使うＤMＡＣはチャネル２）

（５）　ＤMＡＣにＤMA転送先頭アドレス、すなわちＶＲＡM先頭アドレスをセット

（６）　ＤMＡＣのターミナルカウントをセット

（７）　次にあげるＤMＡＣの各モードの設定。ＤMA動作（キャラクタモード）、１行あたりの文字数（80）、ブリンキング時間（カーソルブリンク32画面／32画面、アトリビュートブリンク32画面：96画面）、１画面あたりの行数（20／25）、表示指定（１行おきでなく通常動作）、カーソル表示モード（ブリンク、反転、ブロック）、１文字に割り当てられる走査線の数（10／20／８／16）、垂直帰線の文字数（６、２、７、３）、水平帰線の文字数（34／26）

（８）　アトリビュートモード（トランスペアレント、白黒／カラー）、１行の最大アトリビュート数

（９）　ＤMＡＣのオートロード、タイムカウント停止、チャネル２イネーブル

（10）　ＣＲＴＣの表示再開

ここで、ＣＲＴＣの垂直帰線信号が垂直帰線サイクルからディスプレイサイクルに変わったのを確認したうえで、次の処理に進みます。

（11）　ＣＲＴＣのＩＦ同期コントロールビットをオフ

最後に、画面をクリアしてカーソルをホームポジションに戻します。

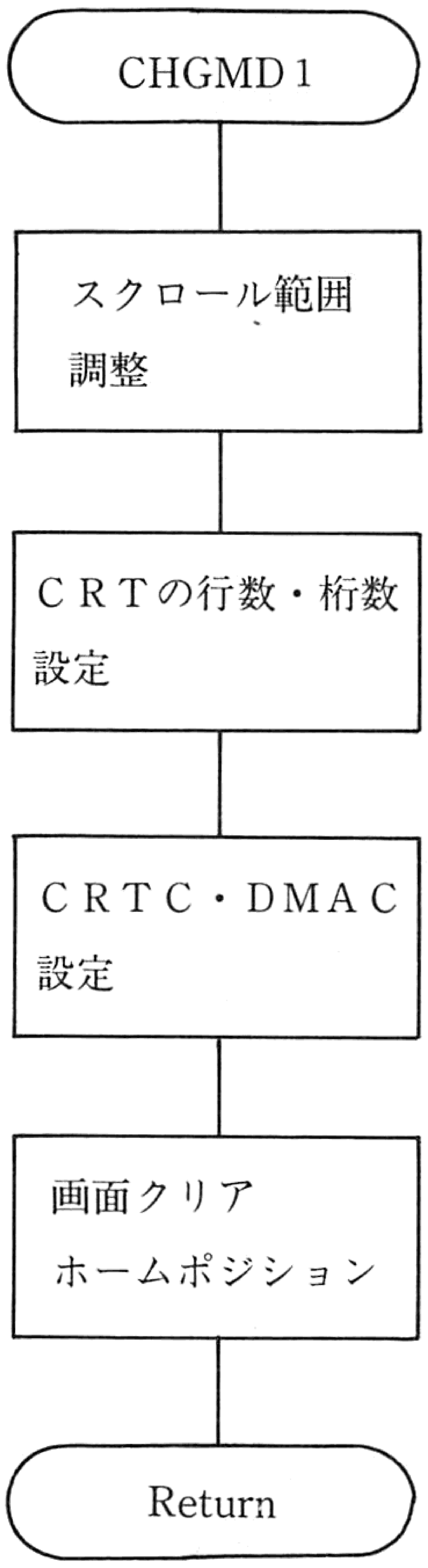


図5－4－3　CHGMD1ルーチン

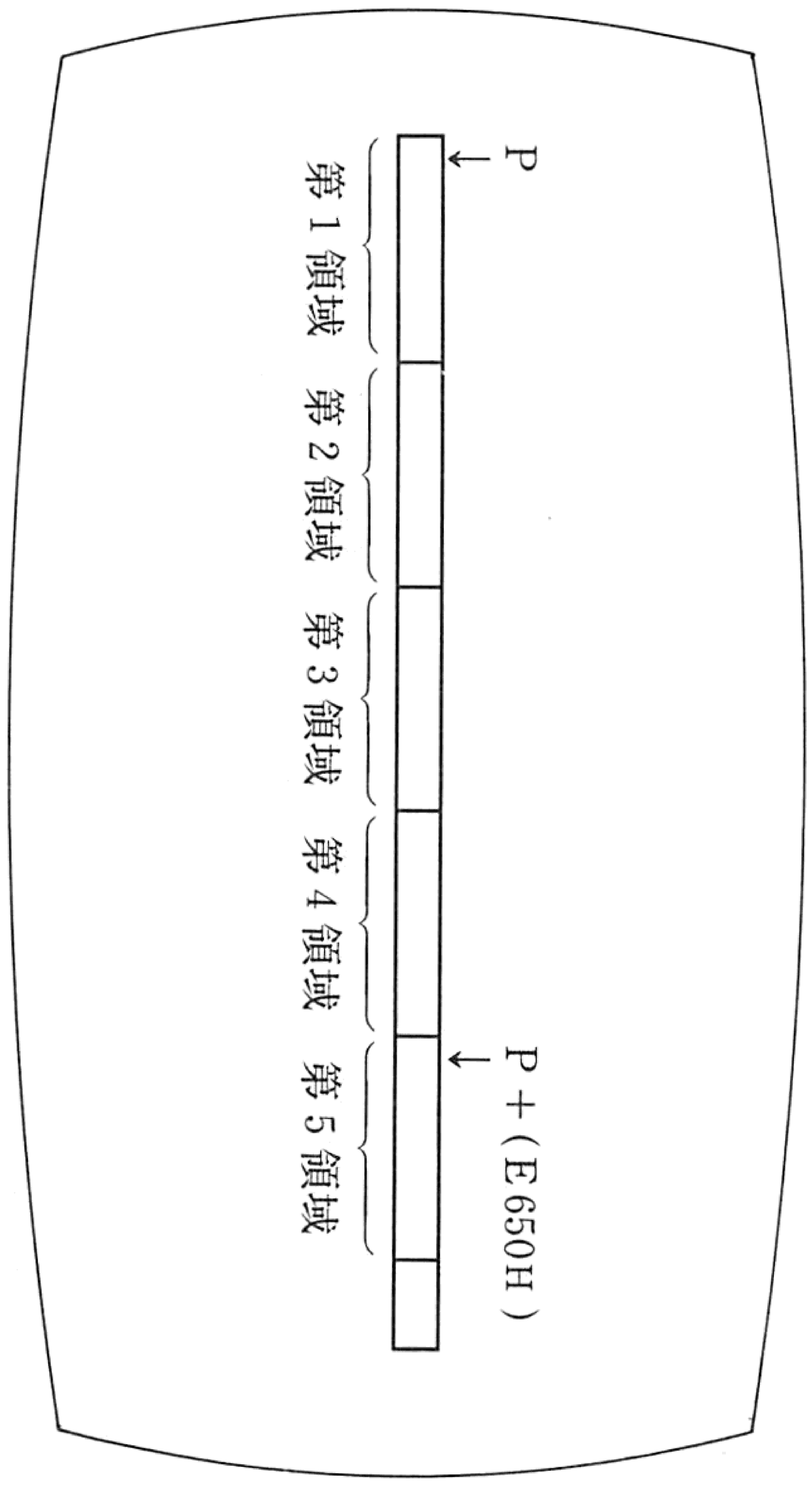


図5－4－4　LINWDTの意味

アトリビュート作成ルーチン：ＭＫＡＴＲ

領　　域	４４５ＢＨ
機　　能	ＣＲＴＣのアトリビュートコードを作成する
破　　壊	Ａ，Ｆ
入　　力	Ｅ６Ｂ９Ｈ（ＣＭＯＤ）：Ｂ／Ｗモードのとき０、カラーモードのときＦＦＨ Ａレジスタ：フォアグラウンドカラーコード
出　　力	Ａレジスタ：求めるアトリビュートコード
ワ ー ク	Ｅ６Ｂ９Ｈ（ＣＭＯＤＥ）：カラーモード
ルーチン	なし

アトリビュートコードの構成は、図５－４－５のようになっています。このルーチンでは、白黒モードの場合には、ＣＭＯＤの下位３ビットのみを取り出し、上位５ビットは０とするだけです。したがって、このルーチンを使っての、グラフィクス・オーバーライン・アンダーラインの指定は不可能です。カラーモードの場合、入力された値の下位３ビットを上位３ビットに移し、ビット３を１に、ビット０、１、２、４を０に設定します。なお、カラーモードにおけるビット４は通常指定とカラー指定との切り換えをします。

ところで、このアトリビュートにおけるグラフィクスというのは、テキストセミグラフィクスのことです。つまり、通常のキャラクタ１文字分の領域を８つに分けてグラフィクスを行おうとするもので、各ビットとエレメントの対応は図５－４－６のようになります。これを利用すると、160×100 ドットのグラフィクス表示が利用可能となります。

画面出力ルーチン：ＣＨＰＵＴ

領　　域	３Ｅ０ＤＨ
機　　能	Ａレジスタのキャラクタを現在のカーソル上に出力する
破　　壊	なし
入　　力	Ａレジスタ：表示するキャラクタ
出　　力	なし
ワ ー ク	Ｅ６ＡＤＨ（ＩＮＳＦＬＧ）：インサートモードフラグ Ｅ６ＢＡＨ（ＬＩＮＥＮＤ）：実際のスクロール終了行 Ｅ６Ｂ５Ｈ（ＮＵＬＣＨＲ）：ヌルキャラクタコード Ｅ６Ｂ６Ｈ（Ｆ．ＬＴＲＬ）：コントロールコード画面出力用フラグ Ｅ６Ｃ１Ｈ（Ｓ．ＳＹＣ１）：ポート４０Ｈに出力したデータ ＥＦ８３Ｈ（ＦＳＴＰＯＳ）：１行入力開始時のＹ座標

(1)モノクロモード

G	0	U	O	0	R	B	S

S----0：通常　1：文字を表示しない（サプレス）
B----0：通常　1：ブリンク
R----0：通常　1：リバース
O----0：通常　1：オーバーラインが追加
U----0：通常　1：アンダーラインが追加
G----0：通常　1：グラフィクス

(2)カラーモード

BL	GR	RD	G	1	0	0	0	通常指定

0	0	U	O	0	R	B	S	カラー指定

RD----0：赤を指定しない　1：赤を指定する
GR----0：緑を指定しない　1：緑を指定する
BL----0：青を指定しない　1：青を指定する
それ以外は，モノクロモードと同じ

図5－4－5　アトリビュートの意味

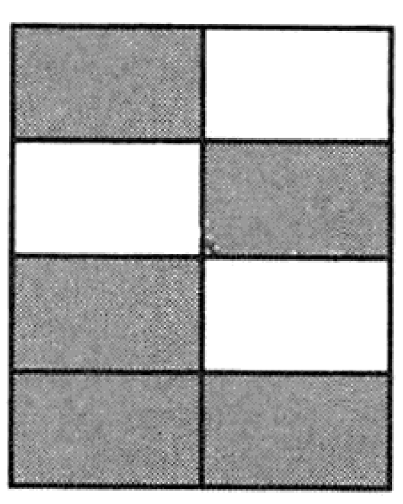
b 0	b 4
b 1	b 5
b 2	b 6
b 3	b 7

キャラクタ

b 7	b 6	b 5	b 4	b 3	b 2	b 1	b 0

データ

例)



1	0	1	0	1	1	0	1	＝　ADH

図5－4－6　テキストセミグラフィクス

ＥＦ８５Ｈ（ＬＳＴＰＯＳ）：１行入力終了時のＸ座標
ＥＦ８６Ｈ（ＣＳＲＹ）　　：カーソルＹ座標
ＥＦ８７Ｈ（ＣＳＲＸ）　　：カーソルＸ座標
ＥＦ８９Ｈ（ＬＩＮＷＤＴ）：画面の桁数
ＥＦ８ＦＨ（ＬＳＴＣＨＲ）：直前表示文字保存用
ＥＦ９ＡＨ（ＬＩＮＴＡＢ）：リンクコードエリア（24バイト）

ルーチン

３Ｅ９ＢＨ：ブザーをならす
３ＥＣ０Ｈ（ＢＵＺＺ２）　：ＢＥＥＰ制御ルーチン
４２８ＢＨ（ＣＵＲＳＯＦ）：カーソルオフルーチン
４４７ＤＨ（ＣＨＯＵＴ）　：１文字出力ルーチン
５ＤＥ５Ｈ：「→」の処理
５ＤＦ５Ｈ：ＣＲの処理
５Ｅ０３Ｈ：ＬＦの処理
５Ｅ２３Ｈ：「↓」の処理
５Ｅ３３Ｈ：「↑」の処理
５Ｅ４９Ｈ：ＨＯＭＥの処理
５Ｅ５４Ｈ：「←」の処理
５Ｆ０ＥＨ：ＣＬＳの処理
５Ｆ７６Ｈ：Ｌレジスタが示す行のリンクコードをＡレジスタに入れる
５Ｆ８６Ｈ：Ａレジスタの値をＬレジスタが示す行のリンクコードとする
６２２０Ｈ：インサートモード時、カーソル以下を１文字ずらす
６４９０Ｈ：行内１文字シフト
６４Ｅ２Ｈ：１行スクロール

最初に、カーソルをオフにします。それから、Ｆ．ＬＴＲＬを調べ、コントロールコードを文字として表示しない（００Ｈ以外の値のとき）場合でＡレジスタの値が２０Ｈよりも小さいときには、コントロールコード処理ルーチン（後述）へと進みます。

コントロールコードの処理でなければ、次にインサートモード時の処理を行います。まず、ＩＮＳＦＬＧを調べて値が００Ｈでない場合、すなわちインサートモードであれば、新たな文字を挿入するためにカーソルに続く文字を１つずつ右へずらしていく必要があります。初めに、ＦＳＴＰＯＳと現在のＹ座標を比較し、一致していれば、ＬＳＴＰＯＳをインクリメントします。ただし、この値がＬＩＮＷＤＴよりも大きくなってしまう場合には、ＬＩＮＷＤＴの値に変更します。この後、カーソルに続く文字を同一行内で右へシフトし、カーソル上を空白にします。さて、ＬＩＮＴＡＢの24バイトには、画面上のリンク情報が入っています。例えば、画面上の７行めと８行めがつながっているとき、この領域の７番目には００Ｈが入ります（つ

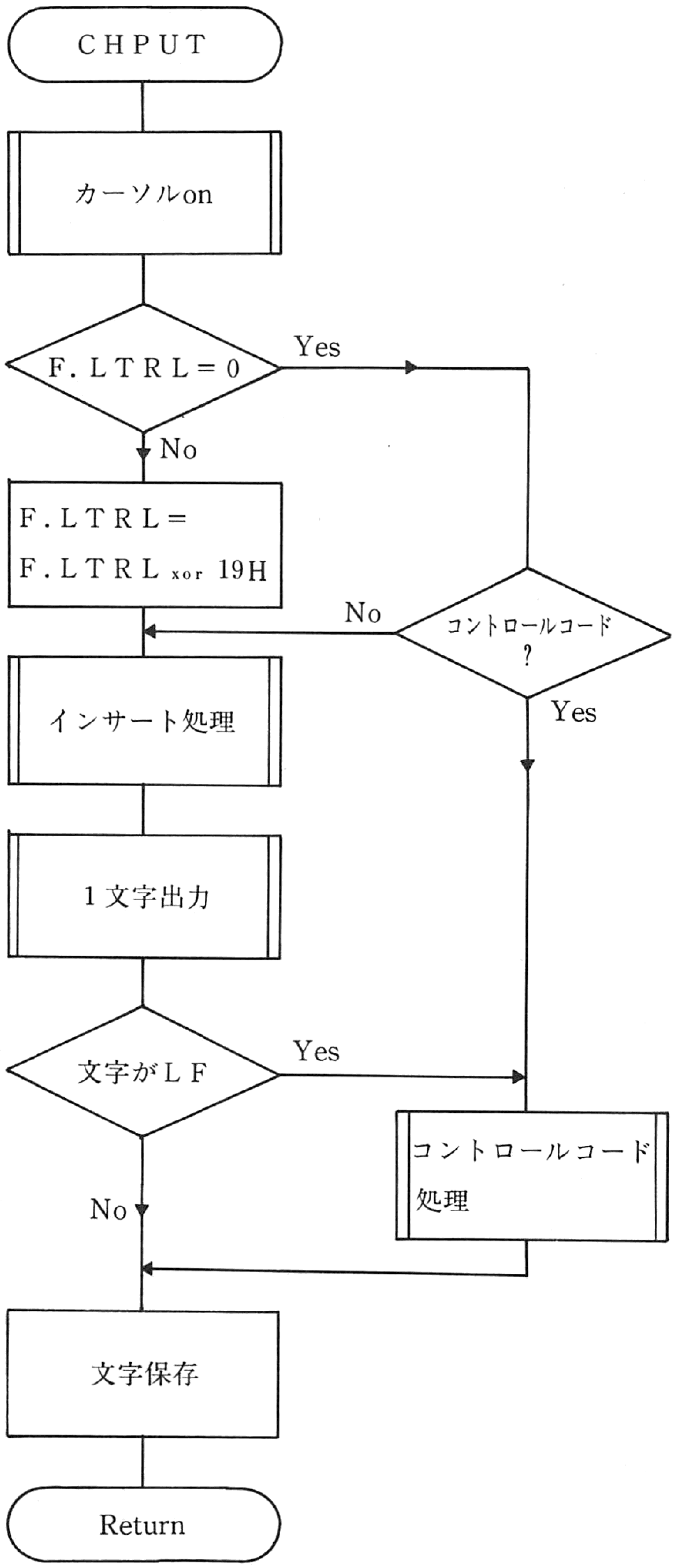


図5−4−7　CHPUTルーチン

ながっていないならば０ＡＨ)。もし、リンク情報を見て、シフトしようとした行と、その後に続く行がつながっているなら、後続の行もシフトしなければなりません。また、シフト前にはつながっていなくても、行末まで文字が一杯詰まっていて、シフトによって意味のある文字(００Ｈ、スペース以外の文字) が次の行にはみ出すような場合には、その１文字のために新たな継続行を設定します。

次に、ＣＨＯＵＴを利用して文字を１つ出力します。このとき、キャラクタコードが０ＡＨ（ＬＦ）であれば、カーソルを行の先頭に設定してコントロールコード処理ルーチンへジャンプします。

それでは、コントロールコード処理ルーチンについて説明します。まず、コードが１９Ｈのときですが、これは次の１文字がコントロールコードである場合、そのコードを文字として画面に出力するためのものです。この処理は、ただ単にＦ．ＬＴＲＬに１９Ｈを書き込むだけです。また、コードを出力した際に常に更新されるＬＳＴＣＨＲは、この場合更新されません。このＣＨＰＵＴルーチンでは、Ｆ．ＬＴＲＬが００Ｈでない場合に、その値と１９ＨのＸＯＲを取ります。そのため、一度Ａレジスタに１９Ｈを入れてＣＨＰＵＴを呼んだ後、別のコントロールコードをＡレジスタに入れて再びＣＨＯＵＴを呼び出すと、そのコードは画面上に文字として現れますが、このとき、Ｆ．ＬＴＲＬはリセットされてしまいますので、それ以降はコントロールコードが文字として現れることはありません。

コントロールコードが０７Ｈの場合にはＢＥＥＰ処理です。これには、ＢＵＺＺ２ルーチンを利用しています。まず、ＢＥＥＰをオンにした後、ソフトウエアタイマに入り、タイマから抜け出したら、今度はＢＥＥＰをオフにします。

その他のコントロールコードの処理は、テーブルによって求めたアドレスに飛ぶことによって行います（図５－４－８)。なお、このテーブルの値は次の式によって求めています。

５ＤＡ５Ｈ＋２×［(Ａレジスタの値)＋０ＥＨ]

また、これらの処理が終わったら、３Ｅ３ＥＨに戻ります。

コントロールコード処理アドレス

０ＡＨ (ＬＦ)	５Ｅ０３Ｈ	※
０ＢＨ (ＨＯＭＥ)	５Ｅ４９Ｈ	
０ＣＨ (ＣＬＳ)	５Ｆ０ＥＨ	
０ＤＨ (ＣＲ)	５ＤＦ７Ｈ	
１ＣＨ (→)	５ＤＥ５Ｈ	
１ＤＨ (←)	５Ｅ５４Ｈ	
１ＥＨ (↑)	５Ｅ３３Ｈ	
１ＦＨ (↓)	５Ｅ２３Ｈ	

※直前の文字が０ＤＨのときは、５Ｅ１７Ｈ

図５－４－８　コントロールコード処理アドレス

ファンクションキー表示ルーチン：ＤＳＦＫＥＹ

領　域	３Ｆ７ＡＨ
機　能	ファンクションキーを表示する
破　壊	Ａ，Ｆ，Ｂ，Ｄ，Ｅ
入　力	Ａレジスタ：０のときＰＦ１～ＰＦ５を、５のときＰＦ６～ＰＦ10を表示
出　力	なし
ワーク	Ｅ６９ＦＨ（ＴＲＭＣＭＤ）：ターミナルモードフラグ
	Ｅ６Ｂ８Ｈ（ＣＮＳＤＦＧ）：ファンクションキー表示フラグ
	Ｅ６ＣＤＨ（Ｆ．ＫＳＵＰ）：キー入力サプレスフラグ
	Ｅ６ＣＥＨ（Ｆ．ＫＹＳＴ）：ファンクションキーフラグ
	ＥＦ８９Ｈ（ＬＩＮＷＤＴ）：画面の桁数
	ＥＦ９０Ｈ（ＣＮＳＤＦ２）：ファンクションキーシフトフラグ
ルーチン	３４Ｄ８Ｈ：ファンクションキーのバッファアドレスを求める
	３４ＥＡＨ：ファンクションキーの文字を取り出す
	４００５Ｈ：Ｂレジスタの値の数だけスペースを画面に表示する
	４０２１Ｈ：ファンクションキーを消去
	４４８ＡＨ：１文字表示

最初に、ＣＮＳＤＦＧを調べます。この値が０のときは、ファンクションキーを表示しません。次に、Ｆ．ＫＳＵＰにＦＦＨを書き込んでキーの入力を禁止します。それから、ＴＲＭＣＭＤを調べて、０以外の値（すなわち、ターミナルモード）となっていて、しかも、表示するファンクションキーがＰＦ６～ＰＦ10である場合には、ファンクションキーのバッファ内の位置を示すポインタ（この時点ではＢレジスタ）を10に設定します（ＰＦ１～ＰＦ５のときは０、ＢＡＳＩＣモードでＰＦ６～ＰＦ10のときは５）。その後、ＬＳＴＩＮＩを用いて、現在のファンクションキーの表示を消去します。

そのあとは、ファンクションキーバッファから文字を取り出して表示していくだけです。まず、行の先頭から２つ（40桁モード）もしくは５つ（80桁モード）スペースを出力します。次に、３４Ｄ８Ｈのルーチンを用いて表示しようとするファンクションキーの文字内容が入ったバッファの先頭アドレスを計算します。アドレスが求められたら、そこから文字を順番に５個（40桁モード）または11個（80桁モード）取り出し、反転文字として画面に出力します。ただし、各文字の前にはスペースが１つずつ加わります(これも白黒反転で)。また、１つのファンクションキーの表示がおわったら、スペースを１つ出力したのち（これは非反転の普通のスペース）、次のファンクションキーの表示へと進みます。すべての表示が終わったら、Ｆ．ＫＳＵＰおよびＦ．ＫＹＳＴをリセットして、このルーチンの終わりとなります。なお、ファンク

ションキーバッファは、Ｅ６Ｆ２Ｈから始まる16(バイト／ファンクションキー)×15(ファンクションキー)＝240バイトの領域で、各文字列の先頭アドレスは、

Ｅ６Ｆ２Ｈ＋16×（ファンクションキーナンバー　－　１）

で計算されます。

ファンクションキー消去ルーチン：ＬＳＴＩＮＩ

領　域	４０２１Ｈ
機　能	ファンクションキーを消去し、画面最終行を初期化する
破　壊	Ａ
入　力	Ｅ６Ｂ８Ｈ（ＣＮＳＤＦＧ)：０にしておく
出　力	なし
ワーク	４２８９Ｈ：120×25の値が入っている
	Ｅ６Ｂ４Ｈ（ＮＵＬＡＴＲ）：ヌルアトリビュートコード
	Ｅ６Ｃ４Ｈ（ＶＲＡＭＡＤ）：ＶＲＡＭ先頭アドレス
	ＥＦ８８Ｈ（ＬＩＮＣＮＴ）：画面の行数
ルーチン	４０３５Ｈ：最終行クリア
	４３１ＥＨ：Ｌレジスタで示される行の先頭アドレスを求める
	４２Ｅ９Ｈ：最終行初期化

このルーチンは、呼ばれた時のキャリーフラグの値によって異なった動作をします（普通は同じ結果となる)。ＬＩＮＣＮＴが画面の最終行となりますので、４３１ＥＨのルーチンを使って最終行の先頭位置のアドレスを求めます。キャリーフラグが"１"の場合には、この最終行をすべてスペースで埋めてしまいます。また、アトリビュートエリアも初期設定します。一方、キャリーフラグが"０"のときは、この行とそのアトリビュートエリアを、ＦＦ８０Ｈからの120バイトのデータで置き換えてしまいます。この領域には、スペースと初期化されたアトリビュートコードが書き込まれてありますので、結局はクリアされてしまうことになるわけです。

カーソルオフルーチン：ＣＵＲＯＦＦ

領　域	４２８ＢＨ
機　能	カーソルの消去
破　壊	なし
入　力	なし
出　力	なし
ワーク	Ｅ６Ａ８Ｈ（ＣＵＲＦＧ２）：カーソル制御のためＣＲＴＣに送るコマンド
ルーチン	なし

ＣＵＲＦＧ２に８０Ｈを書き込みます。この情報は、ＶＲＴＣ割り込みの際にＣＲＴＣに伝えられます。

カーソルオンルーチン：ＣＵＲＯＮ

領　域	４２９０Ｈ
機　能	カーソルの表示
破　壊	なし
入　力	なし
出　力	なし
ワーク	Ｅ６Ａ７Ｈ（ＣＵＲＦＧ）　：カーソル表示フラグ Ｅ６Ａ８Ｈ（ＣＵＲＦＧ２）：ＣＲＴＣに送るカーソル情報
ルーチン	なし

ＣＵＲＦＧを調べて、そのビット０が"１"であれば、ＣＵＲＦＧ２に８１Ｈを書き込みます。この値は、ＶＲＴＣ割り込みの際にＣＲＴＣに送られて、カーソルが表示されるようになります。

ＶＲＡＭアドレス計算ルーチン：ＶＡＤＤＲ

領　域	４２９ＤＨ
機　能	与えられたスクリーン位置のＶＲＡＭのアドレスを求めます
破　壊	なし
入　力	Ｈレジスタ：桁の位置（１から数えて） Ｌレジスタ：行の位置（１から数えて）

出　　力	ＨＬレジスタ：求めるアドレス
ワ　ー　ク	Ｅ６Ｃ４Ｈ（ＶＲＡＭＡＤ）：ＶＲＡＭの先頭アドレス
	ＥＦ８９Ｈ（ＬＩＮＷＤＴ）：画面の桁数
ルーチン	４３１ＥＨ：Ｌレジスタで示される行の先頭位置のアドレスを求める
	４３３３Ｈ：（Ａレジスタ）×120 をＢＣレジスタに入れる

まず、その行の先頭のアドレスを求めます。これは、４３１ＥＨのルーチンが行い、

ＶＲＡＭＡＤ＋［(Ｌレジスタ)－１］×120

で計算されます。この値に［(Ｈレジスタ)－１］を加えると、アドレスが求められます。ところで、120 を掛けるところでは実際に乗算を行わず、テーブルから値を求めています。４２５７Ｈからの 52 バイトには、120×，120×１，120×２，……，120×25 という値が入っています。これを用いれば、画面上の行の先頭アドレスが簡単に求められます（図５－４－９）。

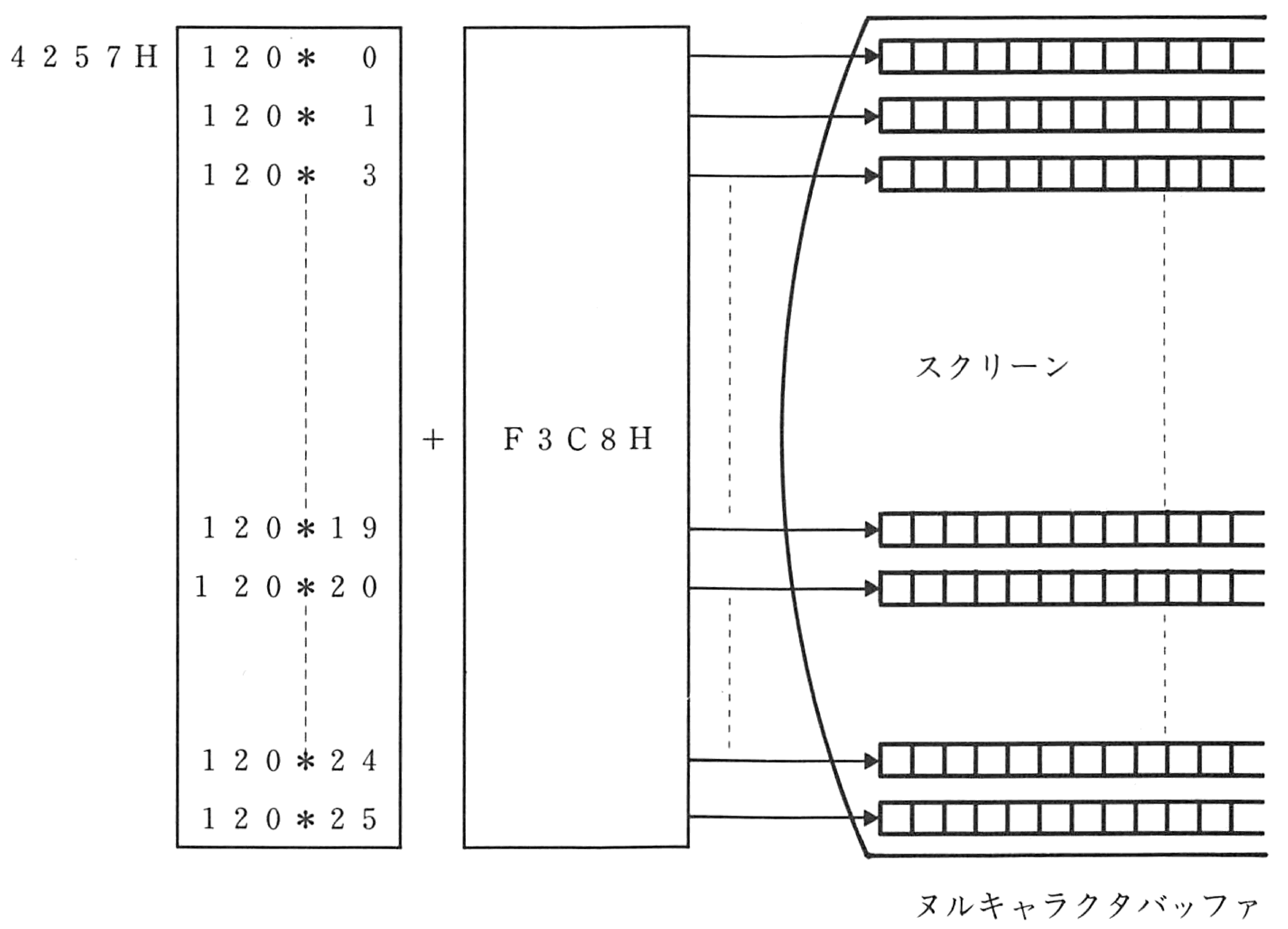


図５－４－９　行の先頭アドレス

ＶＲＡＭゲットルーチン：ＶＲＭＧＥＴ

領　域	４４５２Ｈ
機　能	ＶＲＡＭからキャラクタコードを得る
破　壊	Ａ，Ｆ，Ｂ，Ｃ
入　力	ＶＲＡＭのアドレス
出　力	Ａ、Ｂレジスタ：キャラクタコード Ｃレジスタ：アトリビュートコード
ワーク	Ｅ６Ｃ４Ｈ（ＶＲＡＭＡＤ）：ＶＲＡＭの先頭アドレス ＥＦ８ＡＨ（ＡＴＲＢＵＦ）：最後に書き込んだアトリビュートコードのアドレス ＥＦ８ＤＨ（ＡＴＲＣＯＬ）：最後に書き込んだアトリビュートコードによって制御される最初の桁
ルーチン	４４２２Ｈ：アトリビュートコードのサーチ

このルーチンの主要部は、アトリビュートコードをサーチする４４２２Ｈからのサブルーチンです。ここでは、最初にＶＲＡＭＡＤとＨＬレジスタが示すアドレスとの差を計算します。そこから、120（＝－ＦＦ８８Ｈ）を引けるだけ引けば、その位置のX座標となります。これに１を加えたものを反転し、さらに82を加えたものを、もとのＶＲＡＭアドレスに加えますと、それがその行におけるアトリビュートエリアの先頭アドレスとなります。あとはアトリビュートエリアのアドレス部とX座標を比較していき、アドレス部の方が大きくなったら、次の１バイトが求めるべきアトリビュートコードであると判断し、このルーチンを終えます。

ＶＲＡＭプットルーチン：ＶＲＭＰＵＴ

領　域	４３５０Ｈ
機　能	ＶＲＡＭ上にキャラクタコードを書き込みます
破　壊	なし
入　力	Ｂレジスタ　：キャラクタコード Ｃレジスタ　：アトリビュートコード ＨＬレジスタ：ＶＲＡＭアドレス
出　力	なし
ワーク	ＥＦ８９Ｈ（ＬＩＮＷＤＴ）：画面の桁数 ＥＦ８ＡＨ（ＡＴＲＢＵＦ）：最後に書き込んだアトリビュートエリアのアドレス

ＥＦ８ＣＨ（ＡＴＲＮＥＷ）：最後に書き込んだアトリビュートコード
ＥＦ８ＤＨ（ＡＴＲＣＯＬ）：最後に書き込んだアトリビュートコードで制御される桁数
ＥＦ８ＥＨ（ＡＴＲＣＮＴ）：最後に書き込んだアトリビュートコードのアトリビュートエリアでの位置

ルーチン　４４００Ｈ：アトリビュートコードの挿入
４４２２Ｈ：アトリビュートコードのサーチ
４３ＣＣＨ：アトリビュートを挿入後、４３ＣＦＨに進む
４３ＣＦＨ：前後のアトリビュートが等しいとき前につめ、行の最終アトリビュートのアドレス部を８０Ｈにする

これから表示する文字のアトリビュートコードをＡＴＲＮＥＷに保存した後、ＣＲＴ上の表示アドレスをキーとしてアトリビュートサーチを行い、その位置の現在のアトリビュートコードを求めます。両者が同じであれば何もしません。異なる場合には、アトリビュートの位置によって次の３つに分かれます。

（１）先頭のアトリビュートでない場合：ＡＴＲＣＯＬから前のアトリビュートが支配する位置を引き、これが1より大きければ（３）に移ります。次に、ＡＴＲＮＥＷと前のアトリビュートを比較し、違うならば（２）に移ります。以上のチェックをパスしたなら、前のアトリビュートの開始位置を現在の位置に変更します。それから、後ろのアトリビュートの開始位置と現在の位置とを比較します。両者が異なるならば、新たなアトリビュートを挿入し、同じであればそれ以降のアトリビュートを詰め、一番最後のアトリビュートをクリアします。

（２）行の先頭である場合：現在のアトリビュートアドレスが後ろのアトリビュートのアドレス部よりも小さいならば、新たなアトリビュートを挿入します。そうでなければ、後ろのアトリビュートをＡＴＲＮＥＷで書きかえます。

（３）以上のいずれでもない場合：現在のアトリビュートを挿入し、（２）へ移ります。ただし、このときＡＴＲＣＮＴを調べ、この値が0であれば、すなわちアトリビュートエリアが一杯のときには、最後のアトリビュートが無視されます。

（１）、（２）、（３）の処理の後、前後のアトリビュートで等しいところがあれば、それらを詰め、最終アトリビュートのアドレス部を８０Ｈにセットして、このルーチンでの処理を終えます。

画面Ｉ文字入力ルーチン：ＧＥＴＳＣＲ

領　域	４４７２Ｈ
機　能	スクリーンから１文字取り出す
破　壊	Ａ，Ｆ，Ｂ，Ｃ
入　力	Ｈレジスタ：桁の位置（１から数えて）
	Ｌレジスタ：行の位置（１から数えて）
出　力	Ａ，Ｂレジスタ：キャラクタコード
	Ｃレジスタ　　：アトリビュートコード
ワーク	なし
ルーチン	４２９ＤＨ（ＶＡＤＤＲ）　：ＶＲＡＭのアドレスを計算
	４４５２Ｈ（ＶＲＭＧＥＴ）：ＶＲＡＭからキャラクタコードを得る

ＶＡＤＤＲを使って座標上のＶＲＡＭのアドレスを計算した後、ＶＲＭＧＥＴを使ってキャラクタコードとアトリビュートコードを求めます。

画面出力ルーチン：ＣＨＯＵＴ

領　域	４４７ＤＨ
機　能	カーソル上に文字を出力する
破　壊	なし
入　力	Ａレジスタ：表示するキャラクタコード
出　力	なし
ワーク	ＥＦ８６Ｈ（ＣＳＲＹ）：カーソルＹ座標
	ＥＦ８７Ｈ（ＣＳＲＸ）：カーソルＸ座標
ルーチン	４２９ＤＨ（ＶＡＤＤＲ）：ＶＲＡＭ上のアドレスを計算
	４３４ＣＨ（ＶＲＭＰＵＴ）：ＶＲＡＭにキャラクタを書き込む
	５ＤＥ５Ｈ：カーソルを１つ進める

ＶＡＤＤＲでＶＲＡＭのアドレスを計算した後ＶＲＭＰＵＴを利用して文字を出力し、カーソルを１つ進めます。

1行入力ルーチン：ＩＮＬＩＮ

領　域	５ＦＣ８Ｈ
機　能	ＣＲ，ＬＦ，ctrl－Ｃを検出するまで入力する
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	なし
出　力	Ｅ９Ｂ９Ｈ（ＢＵＦ）：これ以降に入力文字が置かれる
ワーク	６０９ＡＨ：インサートモードの終了を示すキーのコード（13バイト） ６０Ａ７Ｈ：処理するコントロールキーのコード（16バイト） ６０Ｂ８Ｈ：コントロールコードの処理アドレステーブル（２×16バイト） Ｅ６９ＦＨ（ＴＲＭＣＭＤ）：ターミナルモードフラグ Ｅ６Ａ０Ｈ（ＴＲＭＲＥＭ）：ターミナルモードリモートＢＡＳＩＣプロトコル用フラグ Ｅ６Ａ１Ｈ（ＴＲＭＣＨＲ）：ターミナルモード時にＢＡＳＩＣで入力した文字 Ｅ６ＡＤＨ（ＩＮＳＦＬＧ）：インサートモードフラグ ＥＢ００Ｈ（ＡＵＴＦＬＧ）：ＡＵＴＯモードフラグ ＥＦ８３Ｈ（ＦＳＴＰＯＳ）：１行入力開始時のＹ座標 ＥＦ８５Ｈ（ＬＳＴＰＯＳ）：１行入力字のＸ座標 ＥＦ８６Ｈ（ＣＳＲＹ）：カーソルのＹ座標 ＥＦ８７Ｈ（ＣＳＲＸ）：カーソルのＸ座標 ＥＦ８９Ｈ（ＬＩＮＷＤＴ）：画面の桁数 ＥＦＢ９Ｈ（Ｆ．ＰＩＮＬ）：エディットモードフラグ
ルーチン	００１８Ｈ：文字の出力 ３５８３Ｈ：キー入力 ５Ａ０８Ｈ：入力デバイスから１文字入力 ５３７２Ｈ：ＯＰＥＮのチェック ５Ｆ８６Ｈ：Ａレジスタの値をＬレジスタが示す行のリンクコードとする ６０４２Ｈ：テーブルサーチ ６０４ＡＨ：コントロールキーの処理 ６０７ＡＨ：テーブルにないコントロールコードの処理 ６０Ｄ８Ｈ：ターミナルモード時に入力デバイスから文字を得る ６５２８Ｈ：Ｌレジスタが示す行がすべてＦＦＨのとき、これをＦＦ８０Ｈからの120バイト（アトリビュートエリアも含め）で置き換える ＥＤ１５Ｈ：フック（ＲＥＴ）

フックから戻った後、ターミナルモード時の入力ルーチンを呼びます。現在ターミナルモードで、リモートＢＡＳＩＣプロトコルが『ＥＳＣ.』または『ＥＳＣ<』で、オープンされているのがＣＲＴ、プリンタ以外のとき、指定されている入力デバイスから文字を入力していきます。ここで、入力が０ＡＨ，０ＤＨと続いたとき、０ＤＨは無視され、また、行の先頭の０ＡＨも無効とされます。このサブルーチンから戻ると、ＦＳＴＰＯＳにＣＳＲＹ、ＣＳＲＸの値が書き込まれます。そして、コントロールモードではＬＩＮＷＤＴ＋１が、ＢＡＳＩＣモードではカーソルのX座標がＬＳＴＰＯＳに代入され、Ｆ．ＰＩＮＬには００Ｈが書き込まれます。次に、ＡＵＴＦＬＧが調べられ、００Ｈであれば６５２８Ｈのルーチンを実行します。さきほどのサブルーチンで入力を受け付けていなければ、ここで１文字の入力となります。それから、ＦＳＴＰＯＳとＬＳＴＰＯＳを更新し、コントロールキーの処理をします。ここで、このＩＮＬＩＮルーチンの終わりを示すキーが検出されたときは、Ｆ．ＰＩＮＬを００Ｈにしてこのルーチンから抜けます。ただし、ctrl－Ｃによってこのルーチンを終了するときは、キャリーフラグがセットされます。ＩＮＬＩＮルーチンを終了させるキーが押されなかったときは、入力した文字をエコーバックして、再び文字を入力します。

それではコントロールキー処理ルーチンについて説明します。まず、入力した文字がインサートモードを終了させるコントロールコード(６０９ＡＨからの13バイト、図５－４－１０に示す)かどうか調べます。もしそうであれば、ＩＮＳＦＬＧをクリアします。次に、処理すべきコントロールコードであるかを６０Ａ８Ｈからの16バイトの表で調べ（図５－４－１１参照)、そうであれば６０Ｂ８Ｈからの表によりその処理アドレスを求て分岐します。表にないコントロールコードは、６０８０Ｈからのルーチンで処理しています。

コード	機　　能
0DH	キャリッジリターン
02H	1つ前のワードに戻る
06H	1つ先のワードへ進む
05H	カーソル位置から後ろを消す
03H	実行の中断
0BH	ホームポジション
0CH	テキスト画面クリア
1CH	カーソルを右へ移動
1DH	カーソルを左へ移動
1EH	カーソルを上へ移動
1FH	カーソルを下へ移動
18H	カーソルを行の最後に移す
15H	1行キャンセル

図5−4−10
インサートモードを終了
させるコントロールコード

コード	処理アドレス	キ　ー
09H	61F9H	TAB, CTRL−I
0AH	611BH	CTRL−J
08H	62B5H	DEL, CTRL−H
12H	6215H	INS, CTRL−R
02H	63A0H	CTRT−B
06H	6375	CTRL−F
05H	6347	CTRL−E
03H	61D1	STOP, CTRL−C
0DH	6154	⏎, CTRL−M
18H	630B	CTRL−X
15H	633F	CTRL−U
04H	6361	CTRL−D
F8H	63F2	ROLL UP
F9H	6405	ROLL DOWN
01H	641F	HELP
7FH	62B5	DEL

図5−4−11

コントロールコード

5−5　カセット処理ルーチン

カセットリードスタートルーチン：ＣＲＤＯＮ

領　域	７ＥＤ０Ｈ
ワーク	カセットからの読み込みを開始する
破　壊	Ａ，Ｆ，Ｈ，Ｌ
入　力	Ｆ００９Ｈ（ＣＡＳＳＰＤ）：ＣＡＳ１（1200 bps）のときＦＢＨ，ＣＡＳ２（600 bps）のときＦＡＨ
出　力	なし
ワーク	Ｅ６ＥＤＨ（ＲＳＣＯＭＤ）：ＲＳ−２３２Ｃ使用中フラグ ＥＦ０ＥＨ（ＳＩＮＴＭ）：ポートＥ６Ｈ出力データ Ｆ１５３Ｈ（ＩＮＳＩＳＯ）：入力用ＳＩ／ＳＯフラグ Ｆ１５４Ｈ（ＯＴＳＩＳＯ）：出力用ＳＩ／ＳＯフラグ
ルーチン	３５Ｃ２Ｈ（ＢＲＫＣＫ）：ブレークチェック ７Ｄ６６Ｈ：ＵＳＡＲＴリセット ７Ｄ９５Ｈ：キュー初期化後、ＵＳＡＲＴ割りこみを許可 ７ＦＥＤＨ：Tape read ERROR エントリ

まず、ＵＳＡＲＴの割り込みを禁止したあと、600 bps／1200 bpsの設定を行い、モータを始動させます。ＢＲＫＣＫを使ってブレークキーのチェックをした後、ＵＳＡＲＴの設定をします。ストップビット１ビット、パリティビット無効、キャラクタ長８ビットとし、データの受信ができる状態とします。それから、キューを初期化してＵＳＡＲＴの割り込みを許可します。

カセット１文字入力ルーチン：ＣＡＳＩＮ

領　域	７Ｆ８７Ｈ
機　能	カセットから１文字入力する
破　壊	Ａ，Ｆ
入　力	なし
出　力	Ａレジスタ：入力データ
ワーク	Ｅ６ＢＦＨ（ＥＲＲＣＳＴ）：ＵＳＡＲＴエラー発生フラグ Ｆ００６Ｈ（ＣＡＳＦＬ２）：カセット用ヘッダサーチフラグ
ルーチン	３５Ｃ２Ｈ（ＢＲＫＣＫ）：ブレークチェック

４５Ｆ８Ｈ（ＧＥＴＱ）　：キューから１文字得る
７ＥＦ１Ｈ：ＵＳＡＲＴ設定
７ＦＥＤＨ：Tape read ERROR エントリ

カセットからＵＳＡＲＴを通じて送られてくるデータは、割り込みによってキュー#１に入ります。したがって、データを取り出しキュー#１より行います。データが入力されていなくてキューが空の場合には、入力があるまで待ち続けますが、同時にブレークチェックも行っています。データが得られた場合、ＥＲＲＣＳＴを調べて受信時エラーが発生していないことを確認します。ＣＡＳＦＬ２が"０"となっていてエラーの発生を示しているのであれば、Tape read ERROR とします。ＳＴＯＰキーが検出されたときも同様です。

カセットリード終了ルーチン：ＣＴＯＦＦ

領　域	７Ｆ１５Ｈ
機　能	カセットからの読み込みを終了する
破　壊	Ａ，Ｆ
入　力	なし
出　力	なし
ワーク	なし
ルーチン	７ＤＡ６Ｈ：キューの初期化、ＵＳＡＲＴ割り込みの停止 ７Ｆ２３Ｈ：ＣＭＴキャリアコントロールをスペースとする

ＵＳＡＲＴ割り込みを禁止にしてキュー#１を初期化してから、ＣＭＴにスペースキャリアを送ります。

カセットライト開始ルーチン：ＣＷＲＴＯＮ

領　域	７Ｆ４ＤＨ
機　能	カセットへの出力を開始する
破　壊	Ａ，Ｆ
破　壊	Ｆ００９Ｈ（ＣＡＳＳＰＤ）：ＣＡＳ１のときＦＢＨ、ＣＡＳ２のときＦＡＨ
出　力	なし
ワーク	Ｅ６ＣＯＨ（Ｓ．ＳＹＣ０）：ポート３０Ｈに出力したデータの控え
ルーチン	７ＥＢＡＨ：ＣＡＳ１・ＣＡＳ２のセット、モータ始動 ７Ｄ６６Ｈ：ＵＳＡＲＴリセット

7F2AH：CMTキャリアコントロールをマークにする
7F7BH：2.0秒タイマ

まず、CAS1・CAS2の設定を行ってモータをオンにします。次に、USARTのリセットした後、ストップビット2ビット、パリティ無効、キャラクタ長8ビットに定め、送信の状態にした後、3.0秒後にCMTキャリアコントロールをマークキャリアとします。さらに0.5秒たったら、このルーチンからリターンします。

カセット1文字出力ルーチン：CASOUT

領域	7FD0H
機能	カセットに1文字出力する
破壊	A，F，H，L
入力	Aレジスタ：データ
出力	なし
ワーク	なし
ルーチン	35C2H（BRKCK)：ブレークチェック 7FEDH：Tape read ERROR エントリ

ポート21Hのビット1を調べ、USARTがレディ状態になるまで待ちます。その後、ポート20Hに出力しますが、STOPキーが押された場合にはエラーとなります。

カセットライト終了ルーチン：CTWOFF

領域	7F1AH
機能	カセットへの書き込みを終了する
破壊	A，F
入力	なし
出力	なし
ワーク	E6C0H（S．SYC0）：ポート30Hに出力したデータの控え
ルーチン	7F8AH：1.0秒タイマ

モータが止まっているのなら何もしません。まだ回転しているのであれば、1.0秒後にモータを止め、CMTキャリアコントロールをスペースにします。

モータ制御ルーチン：MOTOR5

領　域	7F35H
機　能	モータのオン、オフを制御する
破　壊	A，F
入　力	Aレジスタ：0のとき停止、0以外のとき回転
出　力	なし
ワーク	E6C0H（S．SYS0）：ポート30Hに出力した値
ルーチン	7F2AH：CMTキャリアコントロールをスペースキャリアにする

ポート30Hのビット3を操作した後、CMTキャリアコントロールをスペースキャリアとします。

5−6　RS−232C

5−6−1　通信方式の概要

シリアルインタフェースに使われているコントローラ・LSI：μPD8251AFCは、シンクロナスモード（同期式）とアシンクロナスモード（非同期式）の両方の使い方が可能です。PC−8801が使用する非同期式では、データのビット位置を示すクロック情報は送られてきませんので、転送データの前後にはそれぞれスタートビット、ストップビットというものが付け加えられています(図5−6−1)。データの入力端子はデータがないとき"1"のレベルであるのに対し、スタートビットは"0"のレベルとなっていますので、データを受け取る側では入力信号が"1"から"0"になるのを見計らってデータを取り込むことになります。正確に言うと、入力が"1"から"0"に変わったら、スタートビットのちょうど中間と思われる時点で入力信号のサンプリングを行い、これが"0"であれば（確かにスタートビットであると確認されたら)、これから送られてくるデータビット、パリティビット(データビット中の"1"の個数が偶奇に照らしてデータの誤りを検出するためのビットがパリティビットで、システムによってはない場合もある)、ストップビットの中心位置を決定し、その位置に従って入力端子の値をサンプリングを行って、それぞれのビットの値とします（図5−6−2)。

ところで、スタートビットとパリティビットの長さは1ビット長と決まっていますが、パリティビットの有無とその偶奇、データキャラクタおよびストップビットの長さは、ファイルのオープン時に指定できるようになっています(図5−1−2参照)。そのほかに決めることができるものとしては、XパラメータとSパラメータがあります。Xパラメータを有効にした場合、受信データが受信用バッファの4分の3まで入ると、自動的に通信相手にctrl−Sコードを送り、データ転送の一時停止を要求します。その後データを取り出して4分の1にまで減ったら、ctrl−Qコードを送って転送の再開を要求します。一方、Sパラメータを有効にすると、7ビットモードでのカナの送受信が可能となります。一般にカナ文字を扱うには8ビット必要ですが、SI（Shift In), SO（Shift Out）を使うことによって、7ビットでの交信が実現できます。SIコード（0FH）以後のデータはカナ文字、SOコード（0EH）以後のデータ英数字とみなされわけです。

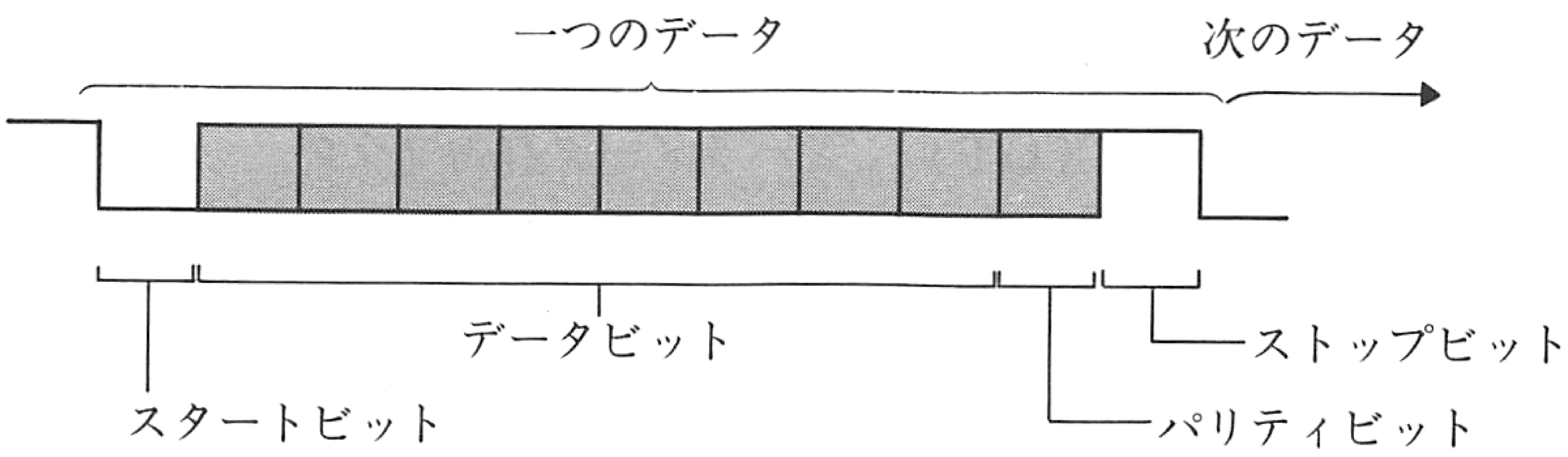


図5－6－1　スタートビットとストップビット

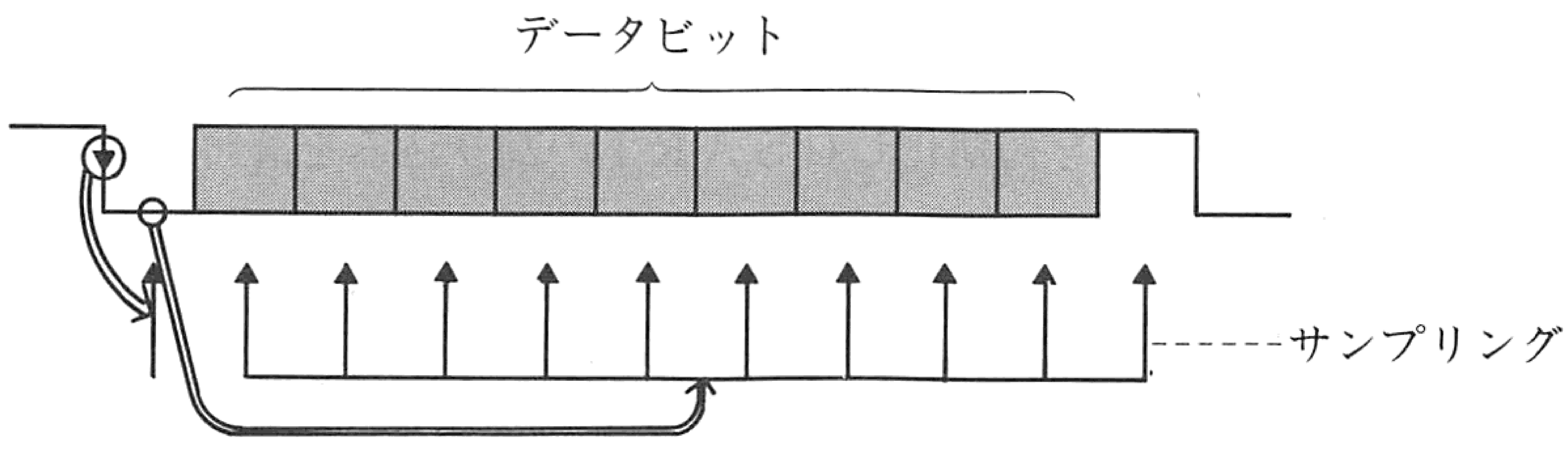


図5－6－2　データのサンプリング

5－6－2　RS－232C処理ルーチン

COM1　チャネルオープンルーチン：COMOPN

領　域	7BC2H
機　能	COM1　チャネルをオープンする
破　壊	A，F，B，C，D，E，H，L
入　力	FILNAM（EC8FH）：BASICのTERMコマンドで使用する"E72X"などのオプションパラメータ RSCOMD（E6EDH）：0 CASACT（E6E8H）：0 HLレジスタ：バッファアドレス DEレジスタ：FE04H
出　力	なし（スタック上の4バイトが取り去られる）
ワーク	EC8FH（FILNAM）：ファイルオープンの際のオプション E6C0H（S．SYC0）：ポート30に出力したデータ E6E8H（CASACT）：カセット使用フラグ

E6ECH（COMSIZ）：COM1用フラグ
E6EDH（RSCOMD）：RS－232C使用中フラグ
F00BH（COMXOF）：Xパラメータ有効フラグ

ルーチン	4DA0H：Bad file name エラー
	4DA3H：File already open エラー
	7A96H：FCB設定
	7B2FH：Eレジスタが80Hのとき、その値を2にかえる
	7CD7H：OPEN時のモード設定
	7D66H：USARTリセット
	7D95H：キューを初期化しUSARTの割り込みを許可する

まず、RSCOMDとCASACTを調べ、カセットまたはRS－232Cが現在使用されていないことを確認します(使われているのならFile already open エラー)。次に、FILNAMからのオープンモードをレジスタに取り込みます。これらの値とCOMSIZにあるWIDTHの値をFCBに書き込みます。その後、いったんUSARTをリセットしてから新たなコマンドをUSARTに送ります。なお、RSCOMDには最後にUSARTに送られたコマンドが保存されます。

カナ送出ルーチン：KANOUT

領域	3203H
機能	カナ文字を SI／SOプロトコルでCOM1に送る
破壊	A，F
入力	Aレジスタ：送信する文字
出力	なし
ワーク	F154H（OTSISO）：RS－232C出力用SI／SOフラグ
ルーチン	3226H（RS2OUT）：RS―232C1文字出力

Aレジスタの値が20Hよりも小さい場合（コントロールコードの時）は、そのまま送信します。次に、OTSISOを調べて、その値が0FH（SI）であって、しかも送り出す文字がカナでない場合には、0EH（SO）を送った後にそのキャラクタを送ります。また、OTSISOがSOで、かつ送り出す文字がカナの場合には、SIをおくった後にそのキャラクタを送ります。SIの状態でカナを送る場合や、SOの状態でカナ以外を送る場合は、そのままキャラクタコードを送るだけです。なお、3226H以降は、次に述べるRS－232C1用の文字出力ルーチンです。

ＲＳ－２３２Ｃ　１文字出力ルーチン：ＲＳ２ＯＵＴ

領　域	３２２６Ｈ
機　能	ＲＳ－２３２Ｃに１文字送る
破　壊	なし
入　力	Ａレジスタ：送り出すキャラクタコード
出　力	なし
ワーク	なし
ルーチン	３５Ｃ７Ｈ（ＢＲＫＣＫ）：ＳＴＯＰキーのチェック

ポート２１Ｈのビット２を調べ、その値が"１"だったら（ＵＳＡＲＴへの出力が可能）データをポート２０Ｈに送ります。もしも送れない場合には、可能になるまで待ち続けます。ただし、ＳＴＯＰキーを押すことによって抜け出すことができます。

受信文字数計算ルーチン：ＰＯＬＣＯＭ

領　域	７５７０Ｈ
機　能	ＣＯＭ１が受信した文字数を計算する
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	なし
出　力	ＨＬレジスタ：受信した文字数
ワーク	Ｅ６ＣＡＨ（ＩＮＴＦＬＧ）：イベントキー（ctrl－Ｃ、ctrl－Ｓ，ctrl—Ｏ）用フラグ ＥＣ８８Ｈ（ＦＩＬＰＴＲ）：カレントファイルバッファのアドレス
ルーチン	４６５５Ｈ（ＮＵＭＱ）：キュー内の文字数を数える ７４９１Ｈ：キー入力後ファイルクローズ

ＩＮＴＦＬＧを調べて、ctrl－Ｃ（ＳＴＯＰキー）が押されているのであれば、ファイルをクローズします。そうでなければ、ＣＯＭ１受信用のキュー#１の文字数を数えます。

ＲＳ－２３２Ｃ　１文字入力ルーチン：ＧＥＴＣＯＭ

領　域	７Ｃ９ＢＨ
機　能	ＲＳ－２３２Ｃポートから１文字入力します
破　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ

入　　力	Ａレジスタ：１
出　　力	Ａレジスタ：キャラクタ
ワ ー ク	Ｆ００ＢＨ（ＣＯＭＸＯＦ）：Ｘパラメータ有効フラグ
ルーチン	３２０３Ｈ（ＫＡＮＰＵＴ）：ＲＳ－２３２Ｃ１文字送出
	３５Ｃ２Ｈ（ＢＲＫＣＫ）　：ブレークチェック
	４５Ｆ８Ｈ（ＧＥＴＱ）　　：キューから１文字得る
	４６５５Ｈ（ＮＵＭＱ）　　：キュー内の文字を数える
	ＥＥＯＢＨ：フック（ＲＥＴ）

ＲＳ－２３２Ｃ受信用のキューから文字を取り出します。もし、キューが空であれば文字を受信するまで待ち続けます。ただし、ＳＴＯＰキーがおされた場合には終了となります。文字が得られたら、キュー内にまだ残っている文字数を数えます。文字数が64未満でＸパラメータが有効の場合、通信先に対して転送再開要求（ctrl－Q）を送ります。

ＣＯＭ１　チャネルクローズルーチン：ＣＯＭＣＬＳ

領　　域	７Ｃ５７Ｈ
機　　能	ＣＯＭ１チャネルをクローズする
破　　壊	Ａ，Ｆ，Ｂ，Ｄ，Ｅ，Ｈ，Ｌ
入　　力	スタック：ファイルポインタ
出　　力	Ｅ６ＥＤＨ（ＲＳＣＯＭＤ）：０
	スタック上の２バイトが失われる
ワ ー ク	Ｅ６ＥＤＨ（ＲＳＣＯＭＤ）：ＲＳ－２３２Ｃ使用中フラグ
	ＥＣ８８Ｈ（ＦＩＬＰＴＲ）：カレントファイルバッファのアドレス
ルーチン	３５Ｃ２Ｈ（ＢＲＫＣＫ）　：ブレークチェック
	４８３ＤＨ：ファイルバッファのクリア
	７ＤＡ６Ｈ：キュー、ＵＳＡＲＴの初期化とＵＳＡＲＴ割り込みの禁止

最初に、キュー・ＵＳＡＲＴを初期化し、ＵＳＡＲＴの割り込みを禁止します。それから、ポート２１Ｈのビット２を調べて、ＵＳＡＲＴ内に送信データが残っていないことを確認します。データが残っているのであれば、それが送られるまで待ちますが、ＳＴＯＰキーが押されると無視してしまいます。その後、ＲＳＣＯＭＤをクリアしてからファイルバッファもクリアします。

5－7　ディスク

5－7－1　ディスクユニットの概要

ＰＣ－８８０１に接続できるフロッピディスクユニットには、DMAタイプのものとインテリジェントタイプのものとがあります。以下、本体と各ユニットとのデータのやりとりの方法について述べてみます。

（1）ＤＭＡタイプの場合

ＤＭＡＣに転送のためのパラメータを設定した後、ＦＤＣにコマンドおよびそれに付随するパラメータを設定をします。その後、コマンドが実行されてデータが転送されると、ＣＰＵに割り込みがかかります。このとき、割り込み処理ルーチンではフラグを立て、メインルーチンにコマンドの終了を知らせます。これを検知したメインルーチンはＦＤＣから送られてくるリザルトステータスを受信して、これをチェックします（リザルトステータスはコマンドによって決められたバイト数だけ受け取る必要がある）。

（2）インテリジェントタイプの場合

インテリジェントタイプのディスクユニットの場合、ＰＩＯを通じてのハンドシェイクによってデータの送受信をします。これには、以下に示す４本の制御信号を使用します。

ＡＴＮ（ATtentioN）：
　　メインシステムがディスクユニットにコマンドを送るときにセットする
ＲＦＤ（Ready For Data）：
　　相手（データの送信側）にデータを受け取る準備ができたことを知らせる
ＤＡＶ（DAta Valid）：
　　相手（データの受信側）にデータを送ったことを知らせる
ＤＡＣ（Data ACcepted）：
　　相手（データの送信側）にデータを受けとったことを知らせる

これらのやりとりの実際については、図５－７－１に示しました。上から順に、矢印の向きに信号が伝えられていきます。この場合には、一度に１バイトだけデータが送られますが、一度に２バイト送信する高速転送もあります（図５－７－２）。これは前図のデータ送信と異な

り、コマンドを送る場合ですが、両者の違いはＲＦＤ＝“１”の送信の前後にＡＴＮのセット／リセットがあるかないかという点だけです。

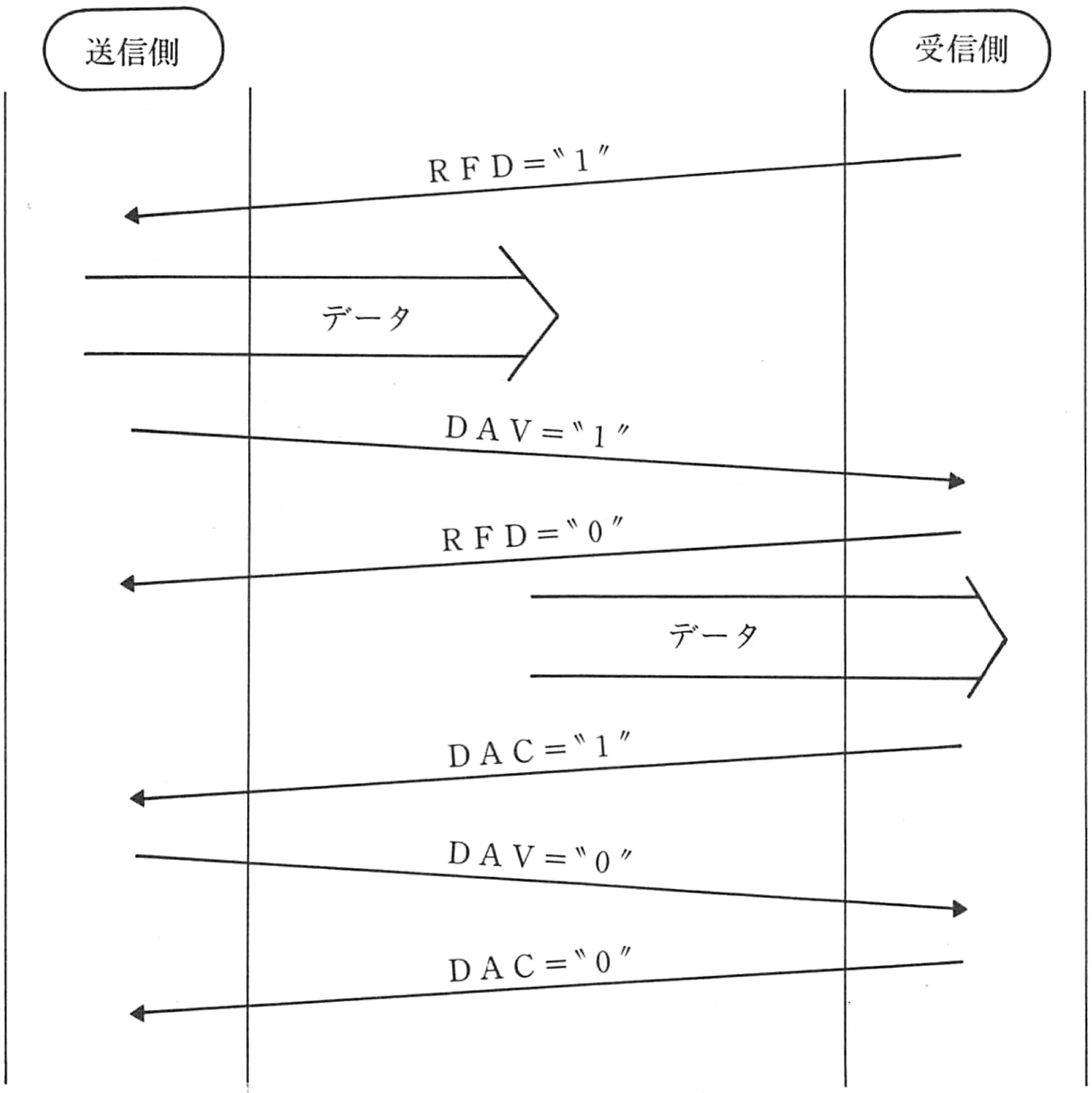


図５－７－１　ディスクユニットとの通信(１バイト)

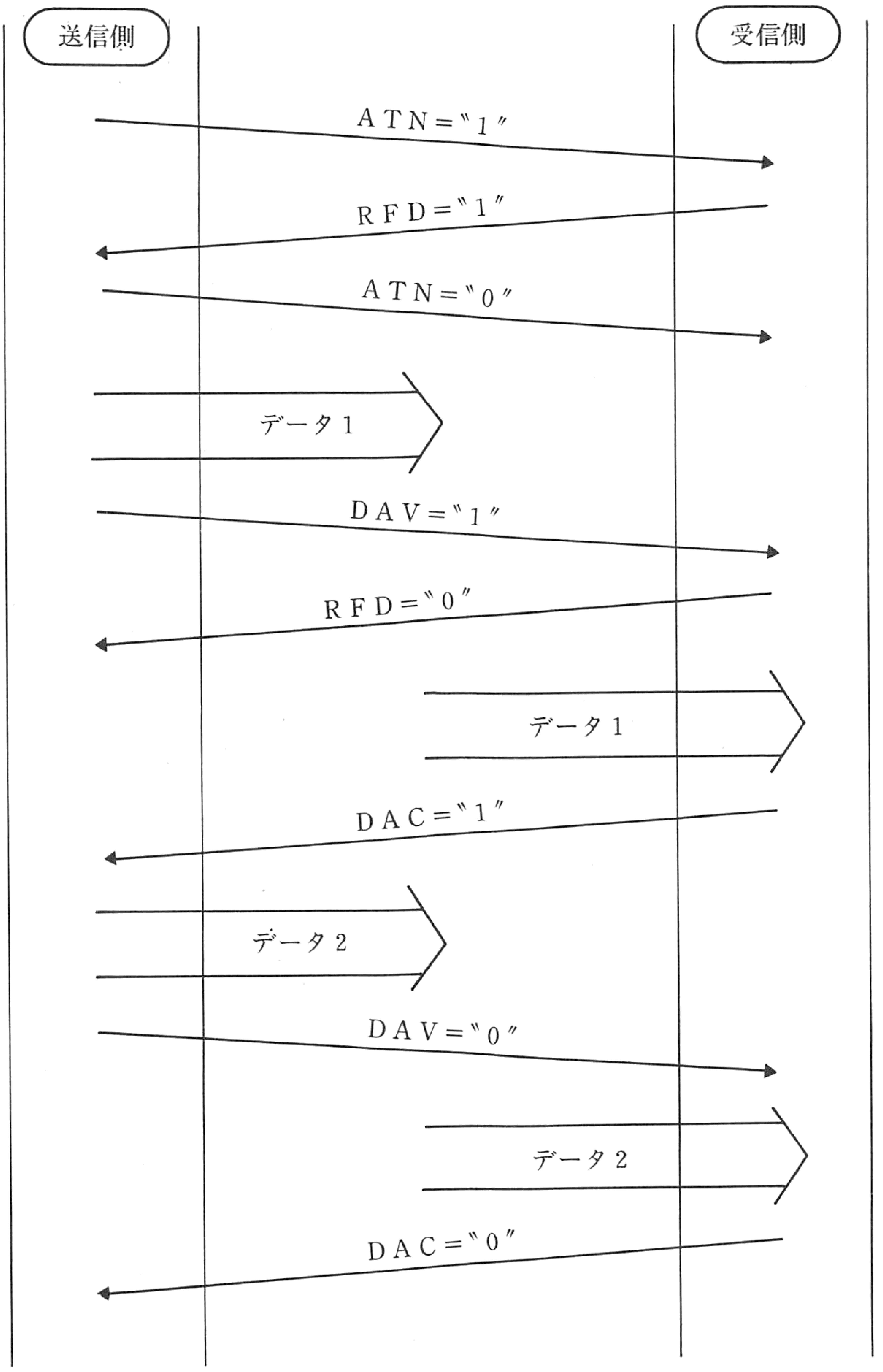


図5－7－2　ディスクユニットとの通信（2バイト）

5－7－2　ディスク処理ルーチン

1セクタ入出力ルーチン：ＰＨＹＤＩＯ

領　域	３６９ＡＨ
機　能	ディスクユニットに対し1セクタのリード、ライトを行う
破　壊	Ａ，Ｆ，Ｄ，Ｅ，Ｈ，Ｌ
入　力	ＥＣ８５Ｈ（ＣＵＲＤＲＶ）：ドライブ番号（0から数えて）
	ＥＦ５ＤＨ（ＣＵＲＴＹＰ）：ドライブのタイプ； 0―DMAタイプ8インチ, 1―DMAタイプ5インチ, 2―片面単密度, 3―両面倍密度
	ＨＬレジスタ：データブロックのアドレス
	Ｂレジスタ：トラック番号（1から数えて）
	Ｃレジスタ：セクタ番号（1から数えて）
	Ｃフラグ　：1のときライト
	Ｚフラグ　：1のときチェック、0のときリード
出　力	Ｃフラグ　：1のときエラー発生
ワーク	ＥＣＢ４Ｈ（ＥＲＲＣＮＴ）：エラーカウンタ
	ＥＦＯＦＨ（ＦＤＩＯＦＧ）：ポートＦ５Ｈへの出力データ
	ＥＦ３ＤＨ（ＥＲＲＳＬ）：DMAディスク用エラー番号
	ＥＦ４ＡＨ（ＮＵＭＳＣ２）：アクセスするセクタ数
	ＥＦ５ＤＨ（ＣＵＲＴＹＰ）：ドライブタイプ
	ＥＦ５ＦＨ（ＤＲＶＮＵＭ）：ドライブナンバー
ルーチン	３６Ｄ６Ｈ：正常復帰
	３７２２Ｈ：インテリジェントタイプディスクコマンド送信
	３７３ＡＨ（ＤＫＲＥＡＤ）：インテリジェントディスク連続データ読み取り
	３７６６Ｈ（ＤＫＷＲＩＴ）：インテリジェントディスク連続データ送信
	３７９０Ｈ：コマンド実行エラーチェック
	３７９ＥＨ：エラーチェック
	３８ＢＤＨ：DMAディスク用ＦＤＣパラメータ設定
	３９７５Ｈ：DMAディスクデータ転送
	３Ｄ６７Ｈ：DMAディスク用パラメータ設定

３Ｄ８ＤＨ：DMA転送データをバッファに書き込む
３ＤＢ７Ｈ：DMA転送データをバッファから得る
３ＤＤ９Ｈ（ＳＥＴＤＲＶ）：デバイスタイプごとのドライブナンバを求める
４２５０Ｈ：タイマ割り込み許可
ＥＥ７４Ｈ：Disk offline エラーエントリ

このルーチンでは、ＳＥＴＤＲＶを使ってタイプ別のドライブ番号を求めた後は、DMAタイプとインテリジェントタイプとでそれぞれ別の処理をします。

インテリジェントタイプの場合：ディスクユニットにリードまたはライトのコマンドを送ったのち，ＤＫＲＥＡＤまたはＤＫＷＲＩＴを実行します。その後、エラーが発生しなかったかどうかチェックします。

DMAタイプの場合：まず、ＦＤＣおよびＤＭＡＣのパラメータを設定します。ライトの場合には、DMA転送用のバッファにデータを書き込みます。次に、タイマ割り込みを許可し、ディスクにデータを転送します。リードの場合には、そのあとバッファのデータを移動します。さらに、ＥＲＲＦＬを調べ、エラーが発生していればそのチェックをします。また、ＥＲＲＣＮＴが4になるとDisk offline エラーとなります。

インテリジェントタイプディスク初期化ルーチン：ＩＮＩＴＤＳ

領　域	３６Ｅ２Ｈ
機　能	インテリジェントタイプディスクの初期化を行う
破　壊	Ａ，Ｆ，Ｄ
入　力	なし
出　力	Ａレジスタ：最大のドライブ番号
ワーク	ＥＣＢ４Ｈ（ＥＲＲＣＮＴ）：エラーの回数
	ＥＦ１４Ｈ（ＤＳＫＴＭＯ）：タイムアウトカウンタ
ルーチン	３６ＤＢＨ（ＤＳＫＣＨＫ）：インテリジェントタイプのディスク接続を調べる
	３７００Ｈ：ディスクドライブの初期化を行い、接続ドライブ数を求める
	３７９０Ｈ：ディスクドライブのエラーチェック

まず、ディスクの接続を調べます。接続されていないのなら何もしません。接続されていれば、最初にハンドシェーク用のＰＩＯをリセットし、それからＥＲＲＣＮＴとＤＳＫＴＭＯをクリアします。その後、ディスクユニットを初期化し、接続ドライブ数を求めます。この値が0である場合には、ディスクユニットにエラーが発生していないかどうか調べてからこのルーチンを出ます。

ドライブタイプ計算ルーチン：ＧＥＴＴＹＰ

領　域	３ＤＣＢＨ
機　能	ドライブ番号からそのドライブのタイプを求める
入　力	Ａレジスタ：ドライブ番号（0から数えて）
出　力	Ａレジスタ：ドライブタイプ（ＰＨＹＤＩＯルーチンと同じ）
	ＥＦ５ＤＨ（ＣＵＲＴＹＰ）：Ａレジスタと同じ
ワーク	ＥＦ５ＤＨ（ＣＵＲＴＹＰ）：ドライブタイプ
	ＥＦ６４Ｈ（ＤＲＶＴＢＬ）：ドライブタイプ対応表；ドライブ0からドライブ11まで12バイトある
ルーチン	なし

ドライブ番号をＮとすると、（ＥＦ６４Ｈ＋Ｎ）が求めるドライブタイプとなります。

タイプ別ドライブ番号算出ルーチン：ＳＥＴＤＲＶ

領　域	３ＤＤ９Ｈ
機　能	各ドライブタイプごとのドライブ番号を計算する
破　壊	Ａ，Ｆ，Ｈ，Ｌ
入　力	ＥＣ８５Ｈ（ＣＵＲＤＲＶ）：ドライブ番号
出　力	Ａレジスタ：タイプ別ドライブ番号
ルーチン	ＥＣ８５Ｈ（ＣＵＲＤＲＶ）：ドライブ番号
	ＥＦ６０Ｈ（ＭＡＸＦＬＰ）：DMAタイプ8インチディスクのドライブ数
	ＥＦ６１Ｈ（ＭＡＸＭＩＮ）：DMAタイプ5インチディスクのドライブ数
	ＥＦ６２Ｈ（ＭＡＸＩＮＴ）：インテリジェントタイプディスクのドライブ数
ルーチン	なし

ドライブ番号からＭＡＸＦＬＰ，ＭＡＸＭＩＮ，ＭＡＸＩＮＴの値を順に引いていき、引けなくなったら、その値が求めるドライブ番号となります。

ディスクドライブチェックルーチン：ＤＳＫＣＨＫ

領　域	３６ＤＢＨ
機　能	インテリジェントタイプのディスクがつながっているか調べる
破　壊	Ａ，Ｆ

入　力	なし
出　力	Ｚフラグ：１のときつながれていない
ワーク	なし
ルーチン	なし

ポート４０Ｈのビット３が"０"であれば、接続されています。

ディスクコマンド送出ルーチン：ＯＵＴＣＭＤ

領　域	３７Ｃ９Ｈ
機　能	インテリジェントディスクにコマンドを送る
ワーク	ＥＦ１４Ｈ（ＤＳＫＴＭＯ）：タイムアウトカウンタ；６×（ＤＳＫＴＭＯ）秒たつとエラーとなる ＥＦ１５Ｈ（ＦＳＴＴＮＳ）：高速転送フラグ ＥＦ５ＤＨ（ＣＵＲＴＹＰ）：ドライブタイプ Ａレジスタ：コマンド
出　力	なし
ルーチン	なし
ルーチン	３７Ｄ２Ｈ（ＯＵＴＤＡＴ）：ディスクデータ送出ルーチン ３ＤＥ９Ｈ：倍速転送設定ルーチン

インテリジェントディスクに送るコマンドが"１"(ライトディスク）または"３"(センドデータ）のとき、それぞれを"１１Ｈ"、"１２Ｈ"に書き換えて高速転送とします（このとき、ＦＳＴＴＮＳも同じ値に書き換えられる)。それから、ハンドシェークポートのＡＴＮを"１"にし、これから送るデータがコマンドであることを知らせます。これ以降は、５―７―７節のＯＵＴＤＡＴルーチンと同じですので、そちらを参照してください。

ディスクドライブデータ送出ルーチン：ＯＵＴＤＡＴ

領　域	３７Ｄ２Ｈ
機　能	インテリジェントディスクにデータを送る
破　壊	Ｆ
入　力	Ａレジスタ：データ
出　力	Ｃフラグ：１のときタイムアウト
ワーク	ＥＦ１４Ｈ（ＤＳＫＴＭＯ）：タイムアウトカウンタ

EF15H（FSTTNS）：倍速転送用フラグ

| ルーチン | 382BH：待ち時間カウントルーチン |

　PIOを通してのハンドシェークによって、インテリジェントディスクにコマンドを送ります。まず、ディスクユニットからの信号RFDおよびDACがそれぞれ“1”、“0”になるまで待ちます（このときタイムカウンタが働く）。それを確認したらATN（ATtentioN）を“0”にし、コマンドを出力します。それが済んだらDAVを“1”にし、ディスクユニットからのDACが“1”になるのを待ちます（このときもタイムカウンタが働く）。次に、FSTTNSを調べて、倍速転送モードであればLレジスタの値も送り出します。その後、DAVを“0”にします。最後に、ディスクユニットからのDACが“0”になるのを待って（このときもタイムカウンタが働く）、このルーチンを終えます。

ディスクデータ受信ルーチン：INDAT

領　域	3847H
機　能	インテリジェントディスクドライブからデータを得る
破　壊	A，F，L
入　力	なし
出　力	Aレジスタ：受信データ Cフラグ　：1のときタイムアウト
ワーク	EF15H（FSTTNS）倍速転送用フラグ
ルーチン	382BH：待ち時間カウントルーチン

　まず、RFDを“1”にしてディスクユニットからのDAVが“1”になるのを待ちます。それが確認できたらRFDを“0”にしてデータを入力します。それから、DACを“1”にし、DAVが“0”になるのを待ちます。次に、FSTTNSを調べて倍速転送モードであれば、再びデータを入力してLレジスタに入れます。最後に、DACを“0”にしてこのルーチンを終えます。

複数セクタリードルーチン：DKREAD

領　域	373AH
機　能	インテリジェントタイプのディスクから複数セクタを同時に読み取る
破　壊	A，F，B，C，D，E，H，L
入　力	Bレジスタ：セクタ数 DEレジスタ：データアドレス

出　　力	なし
ワ　ー　ク	ＥＦ５ＤＨ（ＣＵＲＴＹＰ）：ドライブのタイプ
ルーチン	３７Ｃ９Ｈ（ＯＵＴＣＭＤ）：インテリジェントディスクにコマンドを送る
	３８４７Ｈ（ＩＮＤＡＴ）　：インテリジェントディスクからデータを取り出す

まず、ディスクユニットにコマンド“３”(センドデータ)を送ります。それから、ＣＵＲＴＹＰを調べ、ディスクが片面のものであれば１バイトずつ、両面であれば一度に２バイトずつデータを読み込んでいきます。なお、このルーチンを呼ぶ際には、このルーチンで指定するのと同じセクタ数のデータをディスクユニット内のリードバッファに読み出しておく必要があります。

複数セクタライトルーチン：ＤＫＷＲＩＴ

領　　域	３７６６Ｈ
機　　能	インテリジェントタイプのディスクに複数セクタのデータを同時に書き込む
破　　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　　力	Ｂレジスタ：セクタ数
	ＤＥレジスタ：データアドレス
出　　力	なし
ワ　ー　ク	ＥＦ５ＤＨ（ＣＵＲＴＹＰ）：ドライブタイプ
ルーチン	３７Ｄ２Ｈ（ＯＵＴＤＡＴ）：インテリジェントディスクデータ送出ルーチン

ＣＵＲＴＹＰを調べたうえで、片面か両面かで異なった動作をします。片面の場合には１バイトずつ送りますが、両面の場合には一度に２バイトまとめて送ります。そのため、両面の場合には３７Ｃ９ＨのＯＵＴＣＭＤルーチンなどを使って高速転送モードにしておく必要があります。

パラメータテーブル変更ルーチン：ＣＨＧＰＲＭ

領　　域	３Ｄ６７Ｈ
機　　能	ＤＭＡタイプドライブのパラメータを変更する
破　　壊	Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
入　　力	ＥＦ５ＤＨ（ＣＵＲＴＹＰ）：現在のドライブタイプ
出　　力	ＥＦ４ＢＨ～ＥＦ５ＤＨ：パラメータテーブル
ワ　ー　ク	３０４１Ｈ：ＤＭＡタイプ５インチディスク用パラメータ領域（18バイト）

３０５３Ｈ：DMAタイプ８インチディスク用パラメータ領域（12バイト）
ＥＦ９ＥＨ（ＤＥＶＴＹＰ）：直前にアクセスしたドライブのタイプ
ＥＦ４ＢＨ（ＦＤ０ＦＬ）　：DMAディスク用ドライブステータステーブルアドレス
ＥＦ５ＤＨ（ＣＵＲＴＹＰ）：現在のドライブのタイプ

ルーチン　なし

直前にアクセスしたドライブと、これからアクセスするドライブが同じであれば、何もする必要がありません。異なる場合、適するパラメータをブロック転送します。

トラックシークルーチン：ＳＥＥＫ

領　域　３ＡＡＤＨ
機　能　トラックのシーク
破　壊　Ａ，Ｆ
入　力　Ｃレジスタ：ビット２－ヘッド番号；　ビット０・１－ドライブ番号
Ｄレジスタ：トラック番号
出　力　なし
ワーク　ＥＦ１６Ｈ（ＣＭＤＦＬ）：DMAディスク転送中フラグ
ＥＦ２６Ｈ（ＳＴＯ）　　：ＦＤＣリザルトステータスＳＴ０
ＥＦ２７Ｈ（ＳＴ１）　　：ＦＤＣリザルトステータスＳＴ１
ＥＦ３ＤＨ（ＥＲＲＦＬ）：DMAディスク用エラー番号
ＥＦ３ＥＨ（ＷＡＩＴＦ）：DMAディスク用ウェイトフラグ
ＥＦ５０Ｈ（ＭＸＴＲＫ）：片面当たりの最大トラック数
ルーチン　３Ａ９１Ｈ：パラメータエラー
３Ｃ７１Ｈ（ＩＷＡＩＴ）：ＦＤＣ割り込み待ちルーチン
３Ｃ９４Ｈ（ＷＴＦＤＣ）：ＦＤＣライトルーチン

Ｃレジスタの値が７より大きかったり、Ｄレジスタの値が最大トラック数よりも大きい場合には、パラメータエラー（ＥＲＲＦＬを８１Ｈにする）となります。このチェックの後、シークを行います。ＦＤＣの実行終了は割り込みによって知ることができますので、割り込みがかかるまで待ちます。割り込みがかかったらＳＴ０、ＳＴ１を調べ、エラーが発生していたらＥＲＲＦＬに８２Ｈを書き込んでシークエラーとします。

ＤＭＡアドレス設定ルーチン：ＳＴＤＭＡ

領　域	３Ｂ８０Ｈ
機　能	ＤＭＡのアドレスを設定する
破　壊	Ａ，Ｆ，Ｈ，Ｌ
入　力	ＥＦ２１Ｈ（ＤＭＡＲＷ）：ライトモードは８０Ｈ、リードモードは４０Ｈ ＨＬレジスタ：ＤＭＡアドレス Ｂレジスタ：セクタ数
出　力	なし
ワーク	ＥＦ２１Ｈ（ＤＭＡＲＷ）：ＤＭＡＣ用コマンドバイト ＥＦ５９Ｈ（ＤＭＡＭＤ）：ＤＭＡコマンド ＥＦ５ＡＨ（ＤＭＡＣＨ）：ＤＭＡＣチャネルアドレス ＥＦ５ＢＨ（ＤＭＡＣ）：ＤＭＡＣベースアドレス ＥＦ５ＣＨ（ＩＦＥＮ）：インターフェースボードイネーブルデータ
ルーチン	なし

ＤＭＡタイプディスクのためにＤＭＡＣを設定し、ＤＭＡインタフェースボードをイネーブルにします。

ＦＤＣデータ書き込みルーチン：ＷＴＦＤＣ

領　域	３Ｃ９４Ｈ
機　能	ＦＤＣにデータを書き込む
破　壊	なし
入　力	Ａレジスタ：データ
出　力	なし
ワーク	ＥＦ５４Ｈ（ＦＤＣＳＢ）：ＦＤＣステータスポートアドレス ＥＦ５５Ｈ（ＦＤＣＤＢ）：ＦＤＣデータポートアドレス
ルーチン	なし

ＦＤＣのステータスレジスタを調べて、データの送信が可能となるのを待ちます（ビット６が“０”、ビット７が“１”となればよい）。その後、データを送ります。

ＦＤＣデータ読み取りルーチン：ＲＤＦＤＣ

領　域	３Ｃ７ＦＨ
機　能	ＦＤＣのデータを読み取る
破　壊	Ａ，Ｆ
入　力	なし
出　力	Ａレジスタ：データ
ワーク	ＥＦ５４Ｈ（ＦＤＣＳＢ）：ＦＤＣステータスポートアドレス ＥＦ５５Ｈ（ＦＤＣＤＢ）：ＦＤＣデータポートアドレス
ルーチン	なし

ＦＤＣのステータスレジスタを調べて、データの受信が可能となるのを待ちます（ビット６が"１"、ビット７が"１"となればよい)。その後、データを受け取ります。

ＦＤＣ割り込み待ちルーチン：ＩＷＡＩＴ

領　域	３Ｃ７１Ｈ
機　能	ＦＤＣの割り込みを待つ
破　壊	なし
入　力	ＥＦ３ＥＨ（ＷＡＩＴＦ）：255
出　力	なし
ワーク	ＥＦ３ＥＨ（ＷＡＩＴＦ）：DMA用ウェイトフラグ
ルーチン	なし

ＦＤＣにコマンドを送って、ある動作を要求した場合、その実行終了は割り込みによって知らされます。そこで、割り込みがかかるまで待機するのがこのルーチンです。このルーチンを呼ぶときにはＷＡＩＴＦを255に設定することになっていますが、割り込みがかかることによってこの値が０に変化しますので、この変化をループの中で調べています。

ＦＤＣステータスチェックルーチン：ＳＴＳＣＫ

領　域	３ＡＦ７Ｈ
機　能	ワークエリア内のＦＤＣステータスを更新する
破　壊	Ａ，Ｆ
入　力	なし

出　　力　ＥＦ２６Ｈ（ＳＴ０）：ステータス０
ＥＦ２７Ｈ（ＳＴ１）：ステータス１
ＥＦ２８Ｈ（ＳＴ２）：ステータス２
ＥＦ２９Ｈ（ＣＣＣ）：シリンダ番号
ＥＦ２ＡＨ（ＨＨ）　：ヘッド番号
ＥＦ２ＢＨ（ＲＲ）　：セクタ番号
ＥＦ２ＣＨ（ＮＮ）　：データ数
ワ ー ク　ＥＦ２６Ｈ（ＳＴ０）：ステータス０
ＥＦ２７Ｈ（ＳＴ１）：ステータス１
ＥＦ２８Ｈ（ＳＴ２）：ステータス２
ＥＦ２９Ｈ（ＣＣＣ）：シリンダ番号
ＥＦ２ＡＨ（ＨＨ）　：ヘッド番号
ＥＦ２ＢＨ（ＲＲ）　：セクタ番号
ＥＦ２ＣＨ（ＮＮ）　：データ数
ルーチン　３Ｃ７ＦＨ（ＲＤＦＤＣ）：ＦＤＣリードルーチン

ＦＤＣからステータスを読み出してメモリに書き込んでいきますが、同時にエラーのチェックも行っています。

ヘッドリストアルーチン：ＨＤＲＳＴ

領　　域　３Ａ８８Ｈ
機　　能　ヘッドのリストア
破　　壊　Ａ，Ｆ
入　　力　Ｃレジスタ：ビット２－ヘッド番号、ビット０、１－ドライブ番号
入　　力　ＥＦ３ＤＨ（ＥＲＲＦＬ）：８１Ｈ－パラメータエラー、８２Ｈ－シークエラー
Ｃフラグ　：エラー発生
ワ ー ク　ＥＦ１６Ｈ（ＣＭＤＦＬ）：ＤＭＡ転送中フラグ
ＥＦ３ＤＨ（ＥＲＲＦＬ）：ＤＭＡディスク用エラーフラグ
ＥＦ３ＥＨ（ＷＡＩＴＦ）：ＤＭＡ用ウェイトフラグ
ルーチン　３ＡＣＥＨ：ＦＤＣエラーチェック
３Ｃ９４Ｈ（ＷＴＦＤＣ）：ＦＤＣライトルーチン

Ｃレジスタが７より大きい場合には、パラメータエラーとなります。エラーでなければリストアのコマンドを送ります。その後、ＦＤＣにエラーが発生していないかどうかチェックします。

第6章

割り込み

CONTENTS

6－1　割り込み処理の概要

6－1－1　PC－8801における割り込み

割り込み処理は、PC－8801において非常に重要な役割を果たしています。これは、1世代前のPC－8001では用いられていなかった機能です（ただし、ユーザ側でハードウェアを用意すれば、使用することも可能でした)。割り込み機能を持たないPC－8001では先行入力ができないという欠点がありました。

さて、PC－8801システム側で割り込みを利用して行っている処理は、上記のキー入力のほかに、CMT・RS－232Cのデータ送受信、カーソル点滅、時間計測を必要とするBASICステートメントの処理、DMAタイプのFDD（5・8インチ）のステータス読み込み、さらに、SRではミュージック並列処理があります。

割り込みの要因となるものには、CMT・RS－232C（との送受信に用いられるμPD8251）によるもの、CRTの垂直帰線信号（VRTC＝Vertical retrace）によるもの、DMAタイプFDDによるもの、600Hzクロックによるもの（リアルタイム割り込み)、さらに、SRには24Hzのクロック割り込み（ミュージック割り込み）などがあります。また、3レベル（SRでは2レベル）が標準装備でユーザに開放されています。これらには優先順位があり、割り込みコントローラの接続端子とそれを処理するソフトウェアで決定されます。

割り込み要因（割り込み名称）	割り込み優先順位
CMT・RS232C(シリアルインタフェース)	0
CRT垂直帰線信号（VRTC）	1
600Hzクロック（リアルタイム）	2
{24Hzクロック(ミュージック：INT3)	3
{ユーザ（INT3）※	不確定
ユーザ（INT4）	不確定
ユーザ（INT5）	不確定
DMAFD5インチ（FDINT1）	} 6
DMAED8インチ（FDINT2）	} 6

※INT3はSRの時ミュージック用に，SR以外ではユーザ用に使用

図6－1－1　割り込みの種類

6－1－2　割り込みのハードウェア

ＰＣ－８８０１のＣＰＵ：μＰＤ７８０（Ｚ８０コンパチブル）は３つの割り込みモードを持っています。このうち、ＰＣ－８８０１では「割り込みモード２」を用いています。

このモードは簡単にいうと、それぞれの割り込みレベルに対する分岐先をアドレステーブルに設定しておき、しかも、そのテーブルの先頭アドレスの上位は可変であるというものです。具体的には、Ｉレジスタにテーブルの先頭アドレスの上位バイトを設定し、外部からの割り込み信号とその直後に送られてくる数値（１バイト）を組み合わせることによって、Ｉレジスタを上位バイト、数値を下位バイトとするアドレスからジャンプ先を得て分岐します。したがって、割り込みベクトルテーブルはアドレス下位バイトが０から始まるところから置かなければなりません。

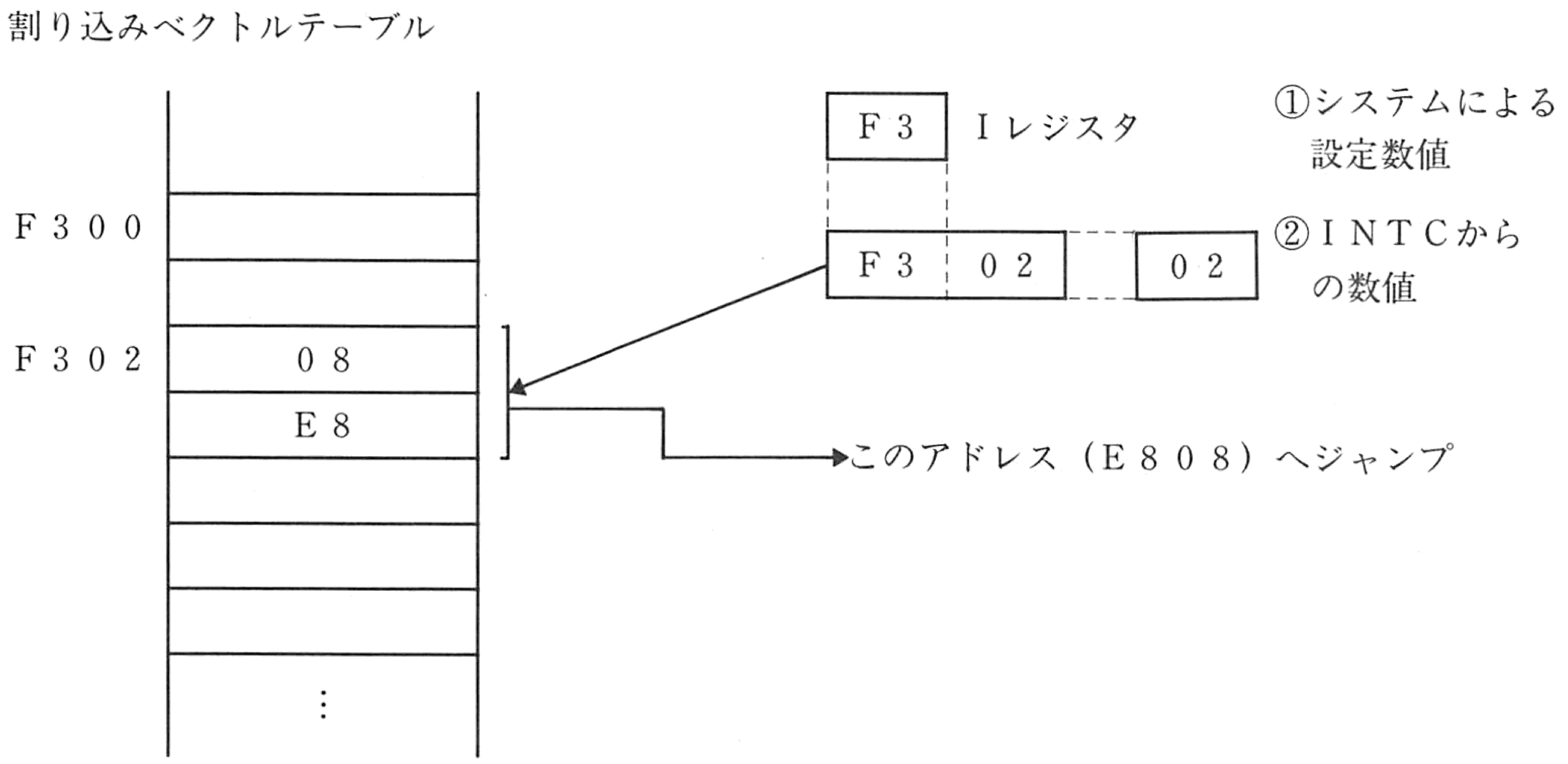


図６－１－２　割り込みモード２（数値はすべて１６進）

デバイスからの割り込み信号から、それに対応する数値を生成し、ＣＰＵへの割り込み信号と数値を同調させて送るという役割を果たすのが割り込みコントローラ（ＩＮＴＣ）です。ＰＣ－８８０１にはＩＮＴＣ：μＰＢ８２１４が１個標準装備されています。このＩＮＴＣは、複数個を接続することによって８レベル以上の割り込み信号を扱うことができるように拡張することができます。

１個のμＰＢ８２１４は８レベルまでの割り込みをコントロールすることができます。それぞれの割り込み信号入力端子には優先順位がつけられており、カレントステータスレジスタ（ＣＳＲ）の値によって割り込みマスクレベルを設定することができます。割り込みマスクレベルとは、それ以上のレベルの割り込みを禁止するための数値です。すなわち、μＰＢ８２１４の割り込み信号入力端子Ｒ０～Ｒ７がそれぞれ第０レベル～第７レベルとなっており、たとえば割り込みマスクレベルが３ならば第３レベル以上の割り込みは禁止されます。ＣＳＲは割り込み処理ソフトウェア側で設定しなければなりません。

ＩＮＴＣの動作をもう少し詳しく見てみましょう。デバイスからレベルＡの割り込み要求があると、ＣＳＲのＳＧＳビット（ＳＧＳはStatus Group Selectの略）をまず参照し、このビットがHighになっているときには、無条件でＣＰＵに対して割り込み信号を出します。つまり、割り込みに優先順位を付けるときには、ＳＧＳビットはＬowでなければなりません。このとき（ＳＧＳがLowのとき）には、ＣＳＲに保持されている値Ｂと割り込みレベルＡを比較し、Ａ＜ＢならばＣＰＵに対し割り込み信号を出し、同時にＡを出力します。Ａ（０～７）は端子Ａ０～Ａ２に出力されます。

ＰＣ－８８０１では、ラッチμＰＢ８２１２を通じ、Ａの２倍の値をＣＰＵに送るようになっています。この値が前述の「割り込み信号直後に送られてくる数値」です。

ＣＳＲへの値の設定は、出力ポートＥ４Ｈに書き込むことで行うことができます。このとき、最上位ビットはＳＧＳビットでビット０～２が割り込みマスクレベルです。

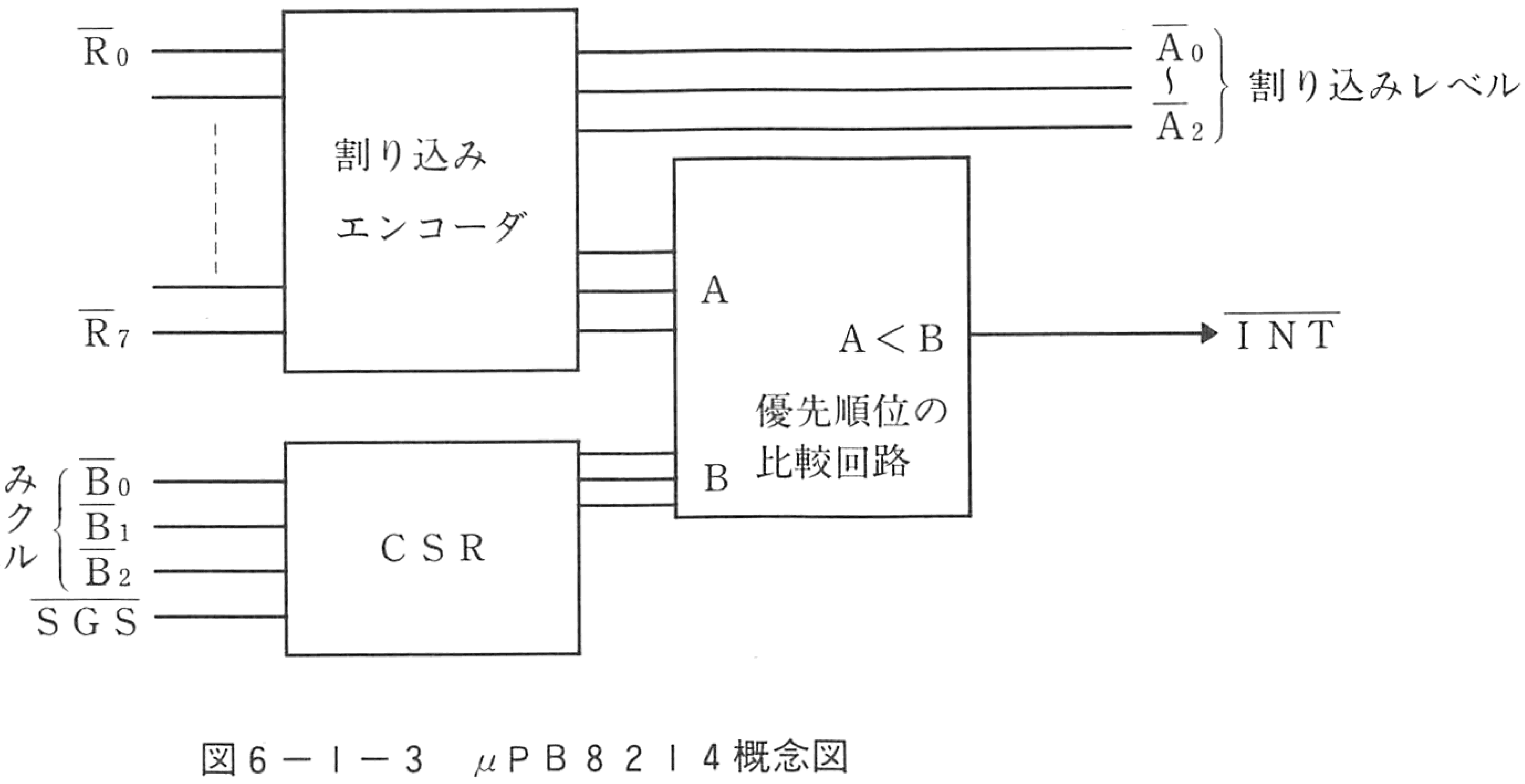


図６－１－３　μＰＢ８２１４概念図

また、他にもうひとつ割り込み要求（シリアルインターフェース、ＶＲＴＣ、クロック）を制御する方法があります。それは、出力ポートＥ６Ｈを用いて行うものです。このポートの第０～第２ビットには、それぞれシリアルインターフェース、ＶＲＴＣ、クロックの各割り込みの許可／禁止を制御するフラグが対応しており、それぞれのビットに“１”を出力したときは許可、“０”を出力したときは禁止となります。

これらの出力ポートはＩＮ命令でその内容を読み出すことができません。そのため、N_{88}－ＢＡＳＩＣではポートＥ４Ｈ，Ｅ６Ｈに出力した内容を、作業域中のＥ６Ｃ３Ｈ，ＥＦ０ＦＨにそれぞれ保存しています。

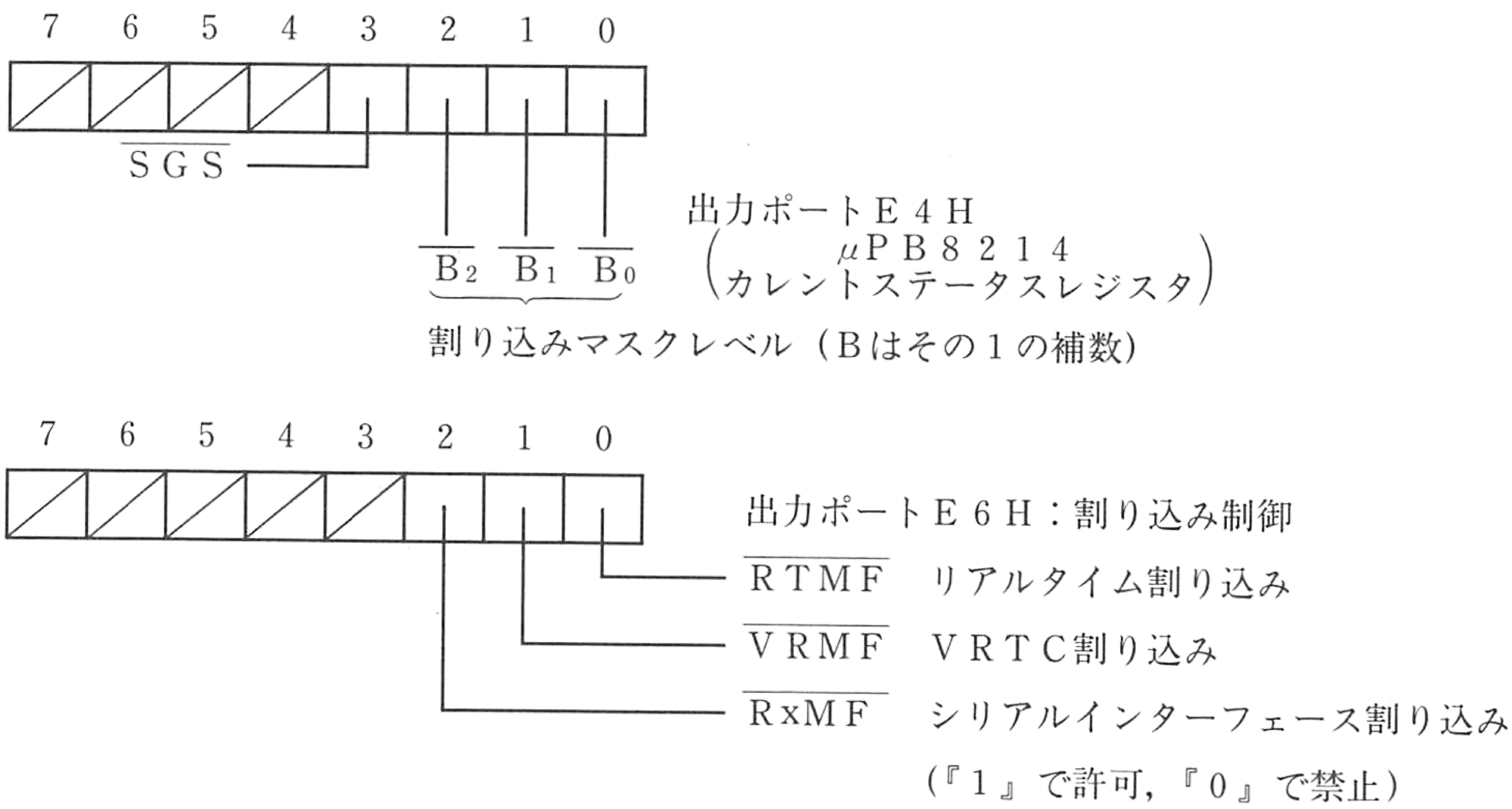


図６－１－４　割り込みコントロールポート

６－１－３　割り込みソフトウェアの概要

ＰＣ－８８０１では、割り込み要求に優先順位を付けています。そのため、ＩＮＴＣのＣＳＲに各々の割り込み処理ルーチンにおいて優先順位に応じた値を書き込まなければなりません。すなわち、自分より優先度が低い（＝自分以上のレベル値の）割り込み要求をマスクするために自分のレベル値をＣＳＲに書き込むのです。ハードウェア的には、このことは割り込み要求信号をラッチ状態から解除するという意味もあります。

割り込み処理ルーチンの概略を図６－１－５に示します。

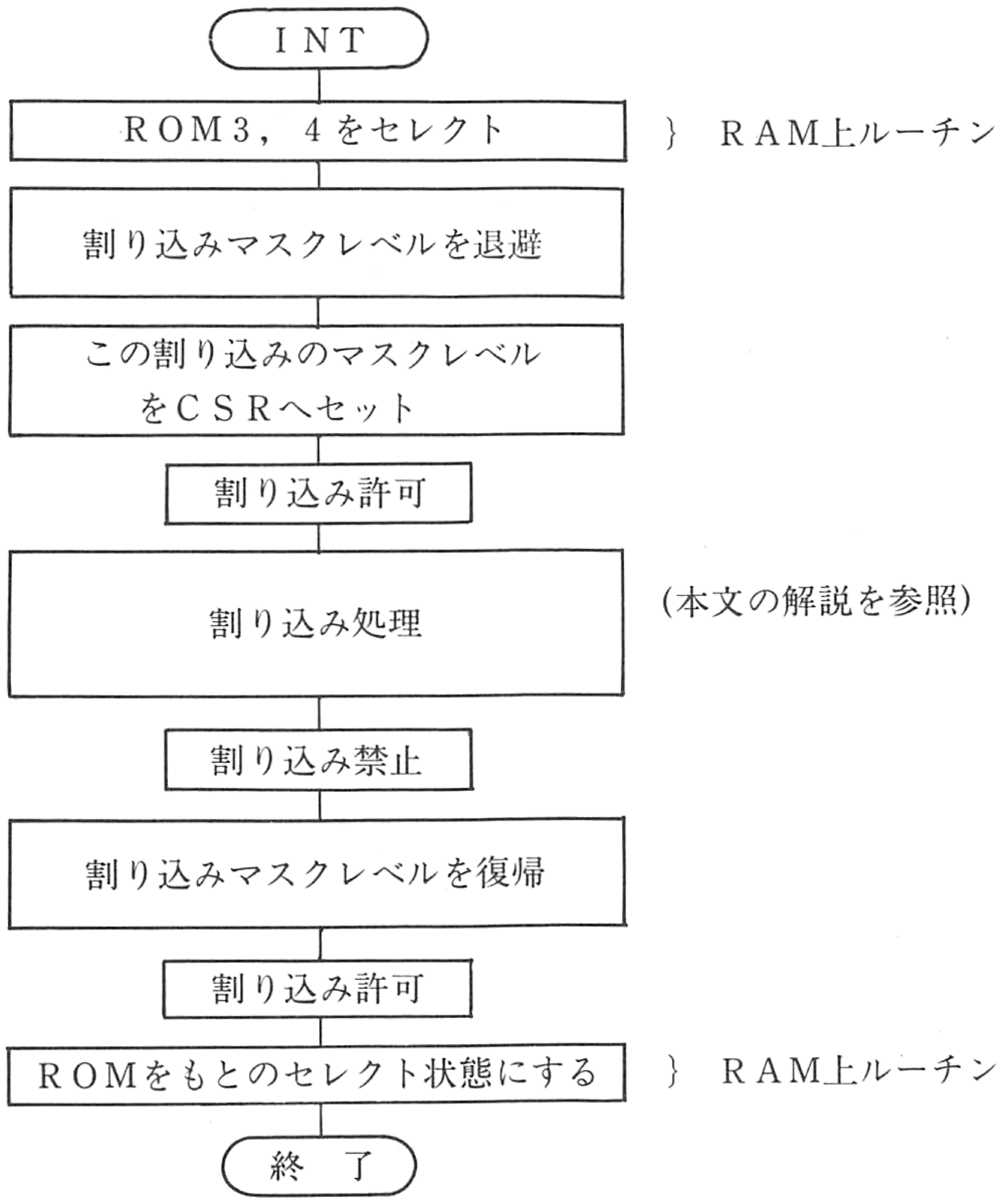


図6－1－5　割り込み処理

ＣＰＵが割り込み信号を受け付けると、(割り込みレベル)×２と割り込みベクトルテーブルとを参照して、ＲＡＭ上にあるＲＯＭ３・４をセレクトするルーチンへジャンプします。これは、割り込み処理の本処理がＲＯＭ３・４上にあるためです。ここでは、ＲＯＭセレクト状態を退避してから本処理へいきます。ＲＯＭセレクトのルーチンは各レベルの処理で共通となっています。

本処理で初めに行うことは、現在行っている割り込み直前の割り込みマスクレベルを退避し、自分の割り込みマスクレベルをＣＳＲにセットすることです。この割り込みマスクレベルの退避は、割り込みの優先順位を確定させるために必要です。

次に、より高い優先順位の割り込みを許可するための割り込み許可命令（ＥＩ）を実行し、各々の機能に従ってその処理を行います。

処理が終わると、割り込みレベルをＣＳＲに復帰し、ＲＯＭセレクト状態をもとに戻して、割り込みを許可してから、ＲＥＴ命令で割り込み処理を終了します。

なお、各処理ルーチンからＣＳＲへ書き込む値は、その割り込みレベル値とは限りません。実際、ＤＭＡタイプＦＤＤのステータスを読み取る割り込み（ＦＤＩＮＴ１・２）では、共にＣＳＲに"６"を書き込み、優先順位では同位となっていますが、割り込みレベルはそれぞれレベル６・レベル７です。また、ＣＳＲは通常のプログラム中でも設定することが可能で、たとえばＣＳＲに０をセットすることで、すべての割り込みを禁止することができます。

各々の処理プログラムの置かれているアドレスは図６－１－６のようになっています。このうち、レベル８～Ｆはテーブルが確保されているのみで、そのレベルの割り込みをコントロールするハードウェアは標準では組み込まれていません。これらのベクトルには実質的に何も行わない割り込み処理ルーチンのアドレス：Ｅ８１４Ｈが書き込まれています。このルーチンの本処理は３０７９Ｈから始まっており、ここでは作業域中にコピーしてある割り込みマスクレベルをＣＳＲに書き込む処理だけを行っています（割り込み要求信号をラッチ状態から解除するためにこの処理は必要です）。

割り込みが発生してから最初に通過するＲＡＭ上のルーチンは、Ｅ７ＥＡＨ～Ｅ８２５Ｈに置かれています。このルーチンのメインはＥ７ＥＥ～Ｅ８０７Ｈで、ここではＲＯＭ３・４をセレクトしてからＨＬレジスタで示されるアドレスをコールし、その後、ＲＯＭのセレクト状態をもとに戻すという作業を行っています。中間の処理は３３９１Ｈの「ＪＰ　(ＨＬ)」命令をコールすることによってＨＬのアドレスをコールし、本処理はサブルーチン形式をとるようになっています。

レベル	テーブルアドレス	処理アドレス	本処理	名　　称
０	Ｆ３００Ｈ	Ｅ７ＥＡＨ	３１６７	シリアル
１	Ｆ３０２Ｈ	Ｅ８０８	３０８０	ＶＲＴＣ
２	Ｆ３０４Ｈ	Ｅ８０Ｅ	４１４３	クロック
３	Ｆ３０６Ｈ	Ｅ８１４※		（ミュージック）
４	Ｆ３０８Ｈ	Ｅ８１４	３０７９	ユーザ
５	Ｆ３０ＡＨ	Ｅ８１４		
６	Ｆ３０ＣＨ	Ｅ８１Ａ	３ＣＢ９	ＦＤＩＮＴ１
７	Ｆ３０ＥＨ	Ｅ８２０	３ＣＡＣ	ＦＤＩＮＴ２
８	Ｆ３１０Ｈ			
９	Ｆ３１２Ｈ			
┆	┆	Ｅ８１４	３０７９	ユーザ
Ｆ	Ｆ３１ＥＨ			

※ＳＲのとき

図６－１－６　割り込みベクトルテーブル

6－2　システム割り込み

6－2－1　シリアルインタフェース割り込み

CMT･RS232Cインタフェースには、μPD8251（USART）が用いられています。このICは、シリアルな信号の送受信を行って、その信号と8ビットのパラレルな信号との変換をとりもつものです。1バイトの送受信データは、μPD8251の内部バッファに作られますが、このとき、バッファ内にデータがある（つまり1文字受信完了した）ということを示す信号がRxRDYであり、これによってシリアルインタフェース割り込みが発生します。このことから、この割り込み処理ではデバイスからデータを受け取り、通信方式によってデータを復元するというのが主な内容となっています。

通信方式には、8ビットキャラクタ方式と7ビットキャラクタ方式があります。7ビットキャラクタの場合は、8ビットのデータがSI／SOコードを用いて7ビットに圧縮されているという形態をとっており、SI／SO処理を行ってデータを復元します。

さて、変換によって得られたデータは受信バッファ内にストアします。受信バッファにはキュー#1が用いられています。キュー#1の大きさは128バイトで、これがいっぱいになったときにはバッファオーバーフローフラグ（E6C9H）をセットします。

一方、受信中にエラーが起こった場合、8251のエラーフラグがセットされるので、このフラグをチェックします。エラーであることが確認されたならば、ワークエリア内のフラグ(E6BFH）を立てて、8251のエラーフラグはリセットします。ただし、CMTのヘッダサーチ中はこれを無視しなければならないので、ヘッダサーチフラグ（F006H）の補数をエラーフラグに書き込み、ヘッダサーチフラグがFFHの場合にはエラーとはみなされないようになっています。

デバイスからのデータ受信において、PC－8801側の処理が追いつかず、バッファがいっぱいになってしまう可能性は十分にあります。このために、Xパラメータを用いてデバイスのデータ送信を中断させることができます。つまり、バッファの空きが64バイト未満となったとき、X－OFF（13H）コードをデバイスに送信し、データ送信を中断させます。この処理は、Xパラメータフラグ（F00BH）がセットされているときにだけ行われ、特にCMTについてはこのようなことはないのでこのルーチンはスキップされます。バッファのデータが処理されて十分空きができ、データ送信を再開させたいときには、X－ON（11H）を送信します。

このあとに、「ON　COM　GOSUB」処理を行って割り込み処理を終了します。もちろん、CMTの場合にはこのルーチンもスキップします。

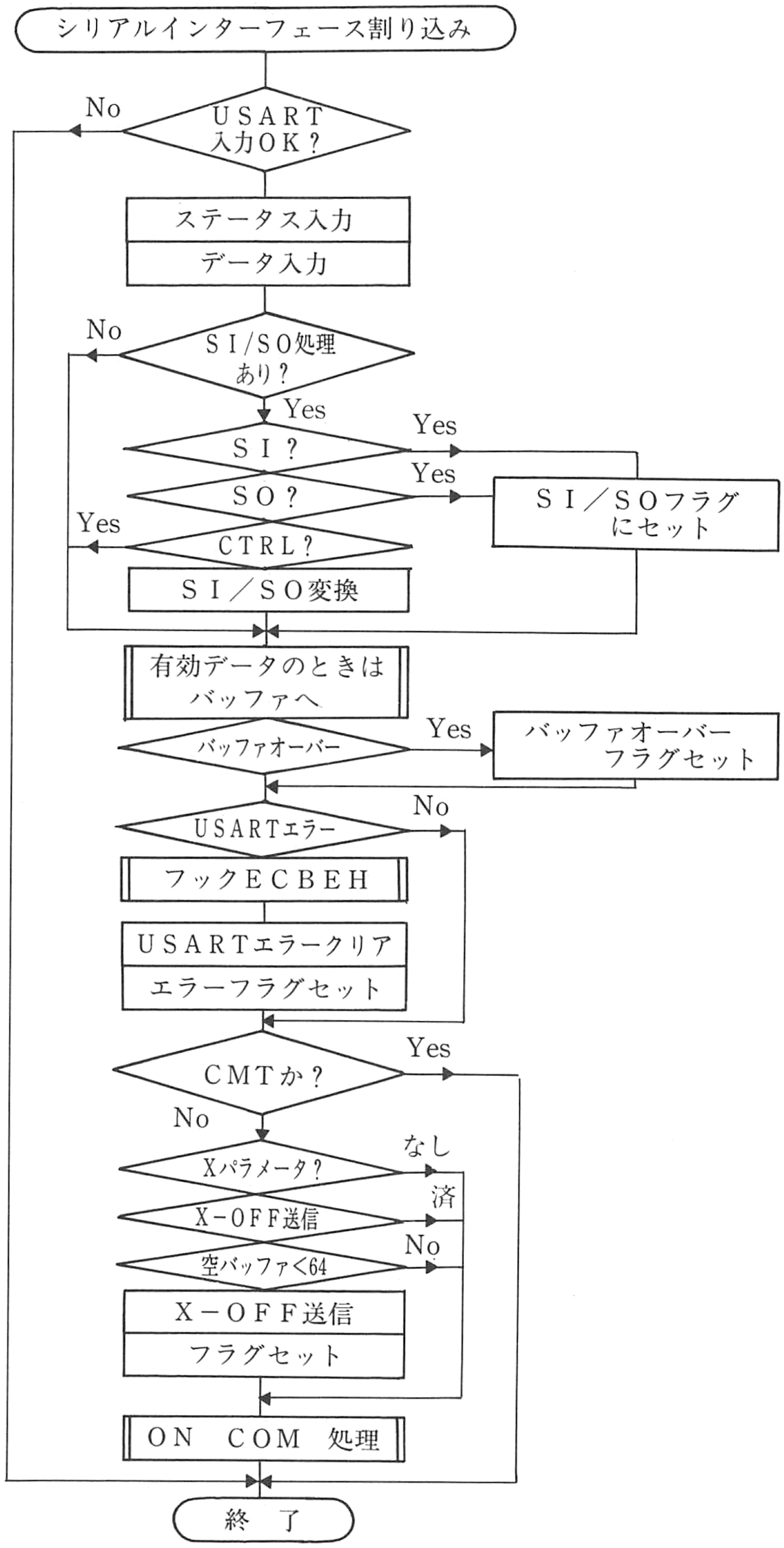


図６－２－１　シリアルインターフェーズ割り込み

6－2－2　ＶＲＴＣ割り込み

ＶＲＴＣ（Vertical retrace）割り込みは、約1／60秒（高解像度ディスプレイモードのときは約1／55秒間隔）のＣＲＴの垂直帰線信号により発生する割り込みです。ＣＲＴ関係の信号で発生するとはいっても、処理内容はＣＲＴ関係とは限らず、むしろリアルタイム割り込みと呼んだ方がふさわしい内容となっています。この割り込みはカーソル表示、ライトペン割り込み、キー入力、ファンクションキー割り込みなどと深く関係しているのです。

カーソル表示については、実際の表示はＣＲＴＣ（μＰＤ３３０１）が行っており、ＣＰＵ側では、ＣＲＴＣにカーソル表示のコマンドを送っているだけです。このとき、作業域中（Ｅ６Ａ８Ｈ）に設定されたデータをコマンドとして送出するため、この値によってカーソルのＯＮ／ＯＦＦを決定することができます（ＯＮには８１Ｈ、ＯＦＦには８０Ｈをセットすればよい）。

ライトペン割り込みは、ハード的にはＰＣ－８８０１の割り込み回路に接続されていません。これは他のキー入力割り込みと同様の方法で処理されます。

以上の処理を済ませた後、キー入力、キー入力割り込みを処理しますが、この直前にキー入力フラグ（Ｅ６ＣＤＨ）をチェックします。したがって、キー入力フラグ≠０とすることによって、これらの処理を行わないようにしておくことができます。

キー入力を受け付けるモードのときに、キースキャン処理（３２４２Ｈ）を呼び、キー入力の有無を調べます。ＰＦキーの入力の有無は、このルーチンによってセットされるキー押下フラグ（Ｅ６ＣＥＨ；入力ありのとき、ゼロ以外の値）で判別することができ、入力があったときには、ＰＦキーNo.に対応するＢＡＳＩＣ割り込みテーブルを参照し、そのキー割り込みフラグをセットします。ＢＡＳＩＣ割り込みを起こさない場合には、ＰＦキーバッファから順次１文字づつ取り出してキー入力バッファ（キュー#０）にセットします。ＰＦキーバッファ１文字取り出しルーチン（３４ＥＡＨ～３５００Ｈ）は、あらかじめＰＦバッファ先頭アドレス（Ｆ００３Ｈ）にセットされたアドレスから終端記号（００Ｈ）までを、次々とＡレジスタに返し、ＣＹフラグ＝１のときに終端であることを示します。

一般のキー入力のときは、オートリピート処理を行います。このため、リピートカウンタ（Ｅ６ＣＦＨ）があり、キー入力変更（それまで押されていたキーとは別のキーが押されたとき）に１ＣＨが設定されます。リピート時には４です。

バッファへのセットは、特殊コードの処理を行うルーチン（３９Ｆ９Ｈ～３１６６Ｈ）を通じて行います。特殊コードとしては、ctrl－Ｃ（ＳＴＯＰ），ＤＥＬ，ＨＥＬＰ（ctrl－Ａ），ctrl－Ｏ，ctrl－Ｓがあります。これらのコードのうち、ＤＥＬを除いて、それぞれの処理を行った後、バッファをクリアし、そのコードをバッファへセットします。しかも、ctrl－Ｃ，ctrl－Ｓ，ctrl－Ｏについては、特殊コード入力フラグ（Ｅ６ＣＡＨ）にそのコードがストアされます。

ctrl－C処理（３１３４～３１５８H）は、ターミナルモード時と通常時の２つに別れます。ターミナルモード時は、ctrl－Cで中断させるか否かを決めるもので、単にＳＴＯＰキーだけが押されているときには無視します。ところが、シフトフラグ（Ｆ００２H）によって「ＳＨＩＦＴ」または、「ctrl」キーが同時に押されていることが検出されたならば、バッファにコード３をセットして中断させます。通常時の処理は、ＨＥＬＰ（ctrl－A）処理（３１５１H～３１６６H）とほとんど同じで、ＢＡＳＩＣ割り込みの有無をテーブルで参照し、割り込みフラグをセットします。

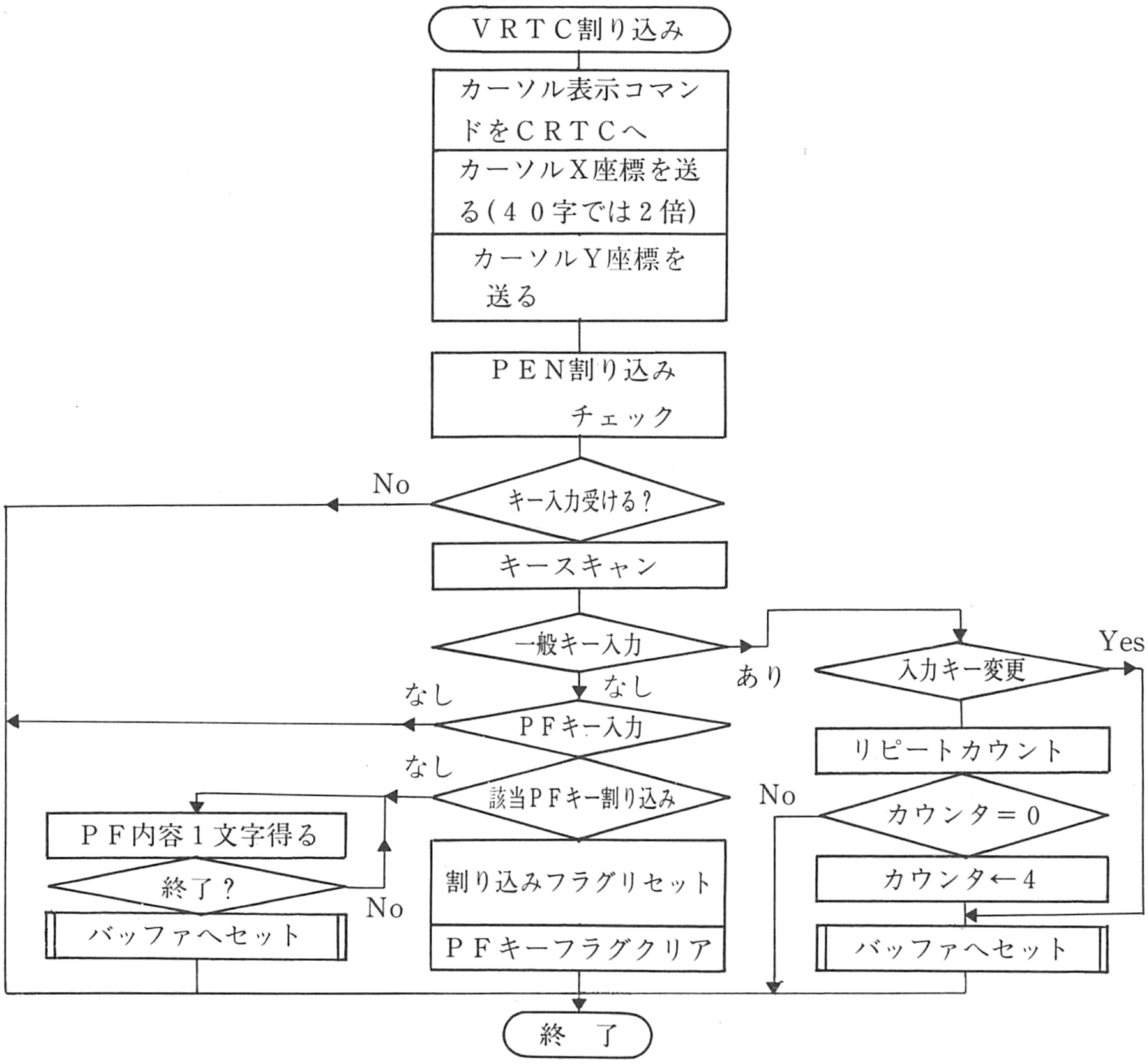


図６－２－２　ＶＲＴＣ割り込み

ＤＥＬ処理（３１２ＡＨ～３１３３Ｈ）は、先行入力をなくすように工夫されています。すなわち、バッファにすでに文字があるときは、そこから最後の１文字を破棄し、バッファが空の場合にのみ、ＤＥＬコード（８Ｈ）をセットします。したがって、ＤＥＬコードが２つ以上バッファに保持されることはありません。しかし、ＤＥＬキー２つでＤＥＬが破棄されることは欠点ともいえます。ただし、この処理はVer 1.1 以降のもので、Ver 1.0 では、ＤＥＬの先行入力が可能になっています。

ctrl－Ｓ，ctrl－Ｏの具体的な処理は、このルーチン内ではいっさい行われておりません。これらについては、先行入力クリアとフラグへのコードセットが行われるだけです。

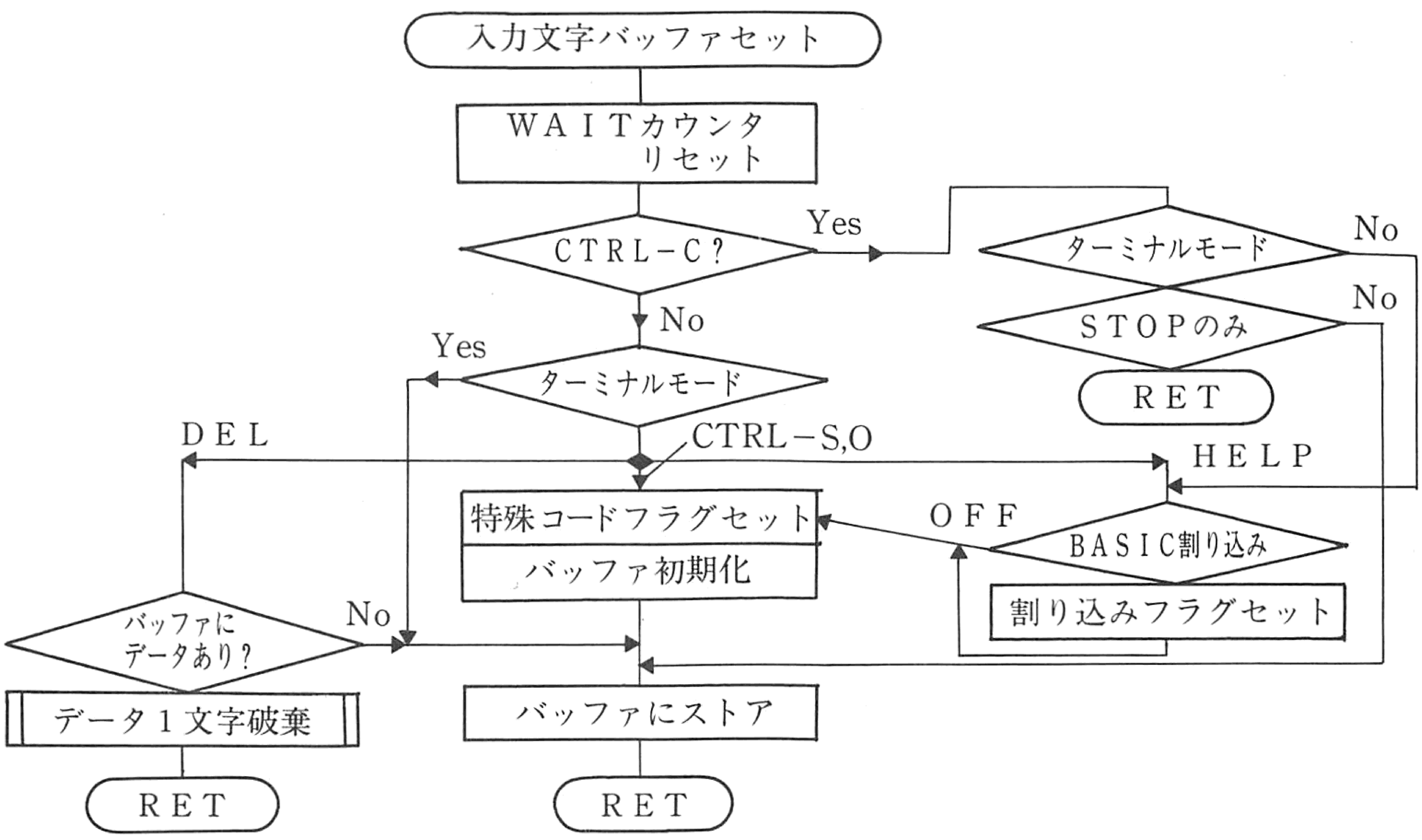


図６－２－３　入力文字バッファセット

６－２－３　リアルタイム（クロック）割り込み

リアルタイム割り込みは１／600秒おきに起こり、カレンダクロック以外の時間計測に使われています。具体的にいえば、キー入力時の制限時間カウント、「ＯＮ　ＴＩＭＥ＄」割り込みのカウント、ディスクモータの動作時間カウントなどがあげられます。

キー入力時の時間制限とは、「ＩＮＰＵＴ　ＷＡＩＴ」のことです。この文によってセットされたカウンタの内容をカウントダウンし、その結果でフラグをセットします。ＩＮＰＵＴ　ＷＡＩＴフラグ（Ｅ６Ｃ６Ｈ）が０でないときにはこの作業を行います。時間のカウントには、「ＩＮＰＵＴ　ＷＡＩＴ」で設定するＩＮＰＵＴ　ＷＡＩＴカウンタ（ＥＦＣ６Ｈ）のほかに、

60進の1次カウンタ（ＥＦＣ８Ｈ）があります。割り込みは1／600秒おきに入るので、キー入力の制限時間は（ＩＮＰＵＴ　ＷＡＩＴカウンタの初期値）× 0.1秒、「ＩＮＰＵＴ　ＷＡＩＴ」では直後の数値を直接カウンタに書き込むので（ＩＮＰＵＴ　ＷＡＩＴで指定した数）× 0.1秒となります。制限時間を超過したことを伝えるフラグには、ＩＮＰＵＴ　ＷＡＩＴフラグが用いられ、この値がＦＦＨになったときに時間超過を示します。また、このときにはキーバッファに1をセットします。

「ＯＮ　ＴＩＭＥＳ＄」割り込みカウント処理は、「ＯＮ　ＴＩＭＥＳ＄　ＧＯＵＳＵＢ」処理中に計算され、セットされた時間をカウントします。このルーチンの構造は、ＩＮＰＵＴ　ＷＡＩＴカウンタとほぼ同じで、異なる点は60進カウンタではなく、1200進カウンタ（正確には1198進、誤差補正のためこのような値となったと思われる）を1次カウンタとして用いている点です。このＢＡＳＩＣ割り込みは秒単位で時間が指定されるので、1200進カウンタではなく、600進カウンタを用いるのが妥当なのですが、そうはしておりません。なぜかというと、600進カウンタでは、2次カウンタは秒数となります。ところが、最大24時間カウントすることを考えると、24時間＝86400秒は2バイトでは収まらず、3バイトの数となってしまいます。2バイトならば、16ビットデクリメント演算があるのですが、3バイトでは相当するものがなく、処理が複雑化します。また、精度の点からも秒単位でとるほど意味がないので1200進を用いるのです。

ディスクモータ動作時間カウント処理は、ＤＭＡタイプ5インチフロッピディスクドライブが最後にアクセスされてから、一定時間以上アクセスされなかった場合に、ドライブのモータを止める処理を行なうものです。ここでは、1次カウンタは600進、2次カウンタは1バイトとなっています。ドライブのモータを止めるには、ポートＦ８Ｈに“0”を出力します。このポートは、モータＯＮ／ＯＦＦの他に書き込み補償のコントロールも行うポートですが、モータＯＦＦ時には書き込み補償ビットを立てることはないので以前の状態を考慮せずに“0”を出力しても問題はありません。

リアルタイム割り込みが必要（割り込ませる必要）か否かを判定するサブルーチン（４１Ｅ３Ｈ～４１ＦＦＨ）が用意されています。これは、ＤＭＡＦＤコントロールポート（Ｆ８Ｈのコピー：ＥＦ０ＦＨ）のモータフラグ、ＯＮ　ＴＩＭＥ＄フラグ（Ｅ６Ｃ７Ｈ）、ＩＮＰＵＴ　ＷＡＩＴフラグ（Ｅ６Ｃ６Ｈ）がカウント中かどうか、ディスクモータ動作時間カウンタがカウント中かどうかをチェックし、このうちひとつでも「ＯＮ」であるか「カウント中」であることが検出された場合には、Ｚフラグに“0”を返します。

リアルタイム割り込み許可（４２５０Ｈ）、禁止（４２４５Ｈ）サブルーチンもあります。これは前述のポートＥ６Ｈの第0ビットをそれぞれ、“1”、“0”にするものです。また、４２４５Ｈは、ＤＭＡＦＤ（5インチ）のモータ停止も行います。

第7章

グラフィクス

CONTENTS

7

7－1　PC－8801におけるグラフィクスの概要

ＰＣ－８８０１のグラフィクス機能は、発表当時の他の機種に比較して非常に優れたものです。その優れた機能としては、ワールド座標系の採用・タイル指定のできるペイント機能・複雑な指定のできるライン機能などをあげることができます。これらの機能は、現在では一般的なものとなりましたが、パーソナルコンピュータの世界に最初に導入したのはＰＣ－８８０１であり、その意味で画期的なものであるといえます。

新しい機能の導入のとき気になるのは、それらによって処理速度が低下しないかということです。特にＰＣ－８８０１の場合、ハードウェア的にも他に比べてあまり有利とはいえず、その上さらに複雑な処理を行うことは好ましくありません。このうち、ワールド座標系については、実際のグラフィクス処理において考慮されており、ワールド座標系を使わないときは余計な処理を行わないようにして速度向上を図っています。速度低下に対する方策は、ワールド座標系以外には何もなされていませんが、メモリ節約のためもあって仕方のないことでしょう。しかし、ＰＣ－８８０１mkIISRにおいては、画面表示の高速化やソフトウェアの改良によってこの点はかなり改善されています。

グラフィクス機能実現のためのアルゴリズムは、ほとんどが一般的なものと極端な違いはありません。代表的なものにLINE文があります。PC－８８０１のLINE文の特徴はラインスタイルを指定できることです。しかしこれは、後に詳しく解説しますが、書き込むドットとしてラインスタイルの各ビットを用いるという簡単な改造によって得ることができます。その逆の例としてPAINT文をあげることができます（そもそも領域の塗りつぶし［以下ペイントという］には、どれも本質的には同じでも、よく知られた一般的なアルゴリズムは見当たりません。そこで、ＰＣ－８８０１などにインプリメントされているアルゴリズムは、これから一般的なものになっていくものと思われます)。ＰＣ－８８０１のPAINT文はタイルペインティングのできる強力なものなので、すでにペイントされている領域とそうでない領域の判定がドットの状態からでは非常に困難なものとなります。それを避けるためには作業領域にペイントした部分を記録することが考えられますが、それでは１プレーン分約16Ｋバイトのメモリが必要になります。これをペイント領域における連結した１ラインの左端点の記録で済ませ、メモリを節約しているのがこのアルゴリズムの最大の特長です。また、このアルゴリズムはさらに改良すると登録された左端点の位置でペイントの終了を判定することができ、改良されたものはＰＣ－９８０１に採用されています。この特長はペイントされる過程を見るとよくわかります。

以上、ＰＣ－８８０１におけるグラフィクス処理の特徴を述べました。それでは、これから具体的に解説することにします。

7－1－1　座標系と画面表示

【1】　画面表示

ＰＣ－８８０１のグラフィクスには、2つの分解能に関する表示モードがあり、ひとつは640ドット×200ライン、もうひとつは640ドット×400ラインです。このために、グラフィクス用として3プレーン＝48Ｋバイトのメモリを持っています。このメモリはＣ０００Ｈ～ＦＦＦＦＨに位置しており、16Ｋバイトずつに分割して割り当てられています。ポート５ＣＨ～５ＥＨにダミーデータを書き込むことでそれぞれプレーン１～３をセレクトします。もとのメモリをセレクトするにはポート５ＦＨにダミーデータを書き込みます。

カラーモードで使われる場合、ドットのメモリ内での位置はアドレスとその１バイト中のビット位置で表すことができます。これに割り当てるパレットの色は3プレーンそれぞれのビットのＯＮ・ＯＦＦの状況で指定されます。また白黒モードの場合には、独立した3画面として使用することができ、メモリ内の位置はカラーモードの場合と同じです。このときにはそれぞれのプレーンが１画面に対応します。さらに、高分解能ディスプレイを用いる場合は、第１プレーンと第2プレーンをはり合わせ、640ドット×400ラインとして使用することができます。すなわち、このときの位置は、プレーン番号・アドレス・ビット位置の３つ組によって指定します。

【2】　座標系

画面表示における数の組とこれらのドットとの対応を座標系といい、ドットを指定するのに用いる数の組をこれらの座標といいます。ＰＣ－８８０１においては次のような座標系が使用されています。

①　基本座標系（オリジナルスクリーン座標系）

前述の通り、グラフィクス画面は640×200（または640×400）個のドットで構成されています。画面上の点の位置が、その点がある行の左端からｘドットだけ右であり、その点がある列の上端からｙドットだけ下であったならば、整数の組〈ｘ，ｙ〉をその点に対応させます。このとき、ｘおよびｙはそれぞれ、0から639、0から199（または399）までの整数をとります。逆に、この条件を満たす数の組で画面上のすべての点を表すことができます。この座標系を基本座標系といいます。ところで、上記の値はあくまで表示される範囲の値であって、座標自身は整数型の許される範囲すべてを指定することができます。

これは、最も基本的な座標系で、組み込みグラフィクス描画ルーチン処理はすべてこの座標系か、これから派生した座標系（後述）に対して描画することに帰着します。

② スクリーン座標系

この座標系は、基本座標系の長方形の部分領域（ビューポートという）に対して、基本座標系と同じ方法で整数の組と座標を対応させたものです。すなわち、上左端点〈a，b〉と下右端点〈c，d〉が与えられたときに、基本座標〈x，y〉とスクリーン座標〈u，v〉（$0 \leqq u \leqq c-a$，$0 \leqq v \leqq d-b$）の対応を、〈x，y〉←〈u＋a，v＋b〉で与えます。

この上左端・下右端のパラメータは、VIEW文によって与えることができます。すなわち、「VIEW（a，b)−(c，d)」を実行すると、上左端点〈a，b〉、下右端点〈c，d〉と設定されます。

スクリーン座標値は基本的に整数型で、そうでない型が現われたときには整数型に変換されます。当然、整数型の表現できる範囲に入らないものは認められません。

③ ワールド座標系

これは物理座標系に対して論理座標系ともいわれ、x座標・y座標とも単精度実数型の値をとることができます(倍精度実数型などは、単精度実数型に変換されます)。画面表示は、このうちの指定された長方形の範囲（ウィンドウという）がビューポート全体に表示されます。このとき、ビューポートとウィンドウの4項点が一致しなければならないので、縦方向・横方向はそれぞれ定数倍されます。

ウィンドウの指定は、WINDOW文を用いて行います。たとえば、「WINDOW（α，β)−(γ，δ)」と指定し、ビューポートが「(a，b)−(c，d)」となっているときには、スクリーン座標〈x，y〉とワールド座標〈u，v〉の対応は次のようになります。

$$\langle x,\ y\rangle \leftarrow \left\langle \frac{c-a}{\gamma-\alpha}\,u,\ \frac{d-b}{\delta-\beta}\,v \right\rangle$$

ここで注意しなければならないのは、ウィンドウ設定の効力は描画コマンドによってすでに描かれたものにはおよばず、ウィンドウが設定された時点から後に実行されたコマンドについて指定されたパラメータを基本座標に変換し、それに対して描画するということです。

上述のように、スクリーン座標は整数型なので、基本座標に変換されたものも整数型でなければなりません。したがって、ワールド座標は変換後に整数型の取り得る範囲内に収まるような値に制限されます。

具体的には、ワールド座標〈u，v〉に対して、

$$-32767 \times \frac{\gamma-\alpha}{c-a} + a \leqq u \leqq 32768 \times \frac{\gamma-\alpha}{c-a} + a$$

$$-32767 \times \frac{\delta-\beta}{d-b} + b \leqq v \leqq 32768 \times \frac{\delta-\beta}{d-b} + b$$

を満たさなければなりません。

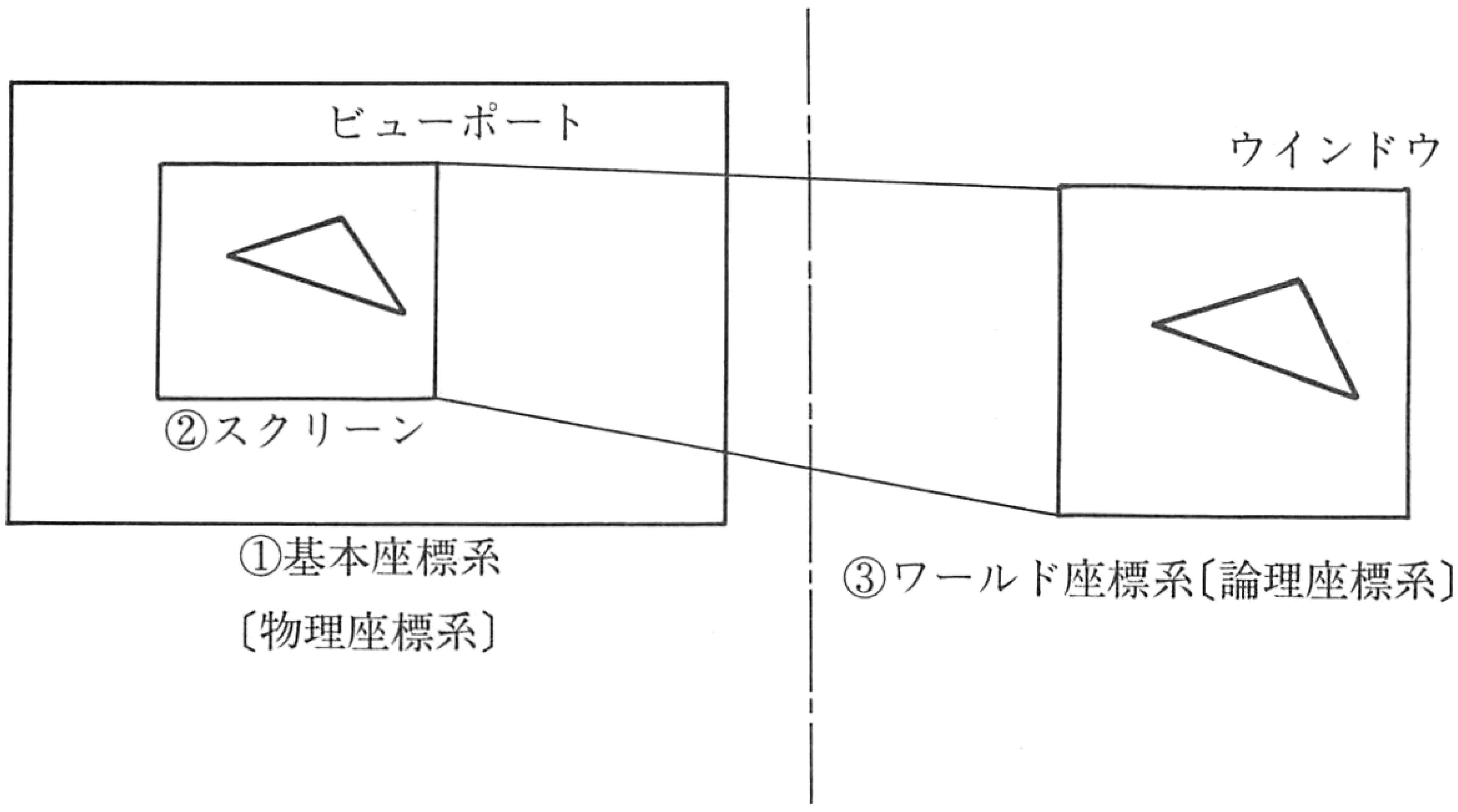


図７－１－１　座標系

④　アドレス・ビット位置（＋プレーン）座標系

［内部のみで使用される座標系（①から派生したもの)］

前にも述べたように、グラフィクス画面は３枚のメモリバンクに分かれており、１ドットはそれぞれ同じアドレスの同じ位置にあるビットに対応しています。したがって、それらの組で基本座標系のすべてのドットを表すことができます。メモリやビット位置など物理的なものに密着しているため、物理座標系ともいわれます。その対応は、基本座標〈ｉ，ｊ〉に対して、アドレス・ビット位置座標〈ａ，ｂ〉は次のようになります。

〈ｉ，ｊ〉←〈(ａ－Ｃ０００Ｈ) mod 80＋（８－ｂ)，(ａ－Ｃ０００Ｈ) ¥ 80〉

(ただし、Ｃ０００Ｈ≦ａ≦ＦＥ７ＦＨ，０≦ｂ≦７)

さらに、400ラインモードのときには,プレーン０とプレーン１をそれぞれ、上200ライン・下200ラインとして用いるため、④にプレーン情報を付け加えればすべてのドットを表すことができます。プレーン情報はそのセレクトポート番号によって表します。

④'プレーン・バイト数・ビット位置座標系

［内部のみで使用される座標系（①から派生したもの)］

これは、プレーン情報はたかだか２ビットで表現できることを利用して、④のアドレスを先頭からのバイト数に変換してアドレス表現に用いる２バイトの上位２ビットにプレーン情報を埋め込むものです(先頭からのバイト数はたかだか14ビットで表現できます)。メモリが節約できるため、これは400ラインモードのときによく使われます。

ちなみに、この座標系は､「ＰＣ－８８０１　N_{88}－ＢＡＳＩＣグラフィクス解析マニュアル」(尾高保　著）においてはスクリーンポインタと呼ばれています。

以上の２つの座標系は、より機械構成に密着したものになっています。しかし、描画の際に行われる計算には明らかに不向きなため、最終的にプロットするときに初めてこの座標系が用いられます。もちろんそうでないものもあり、たとえば、LINE文の機能である長方形の塗りつぶしなどは先に変換してからの方がかなり高速に行えるので、そのようなアルゴリズムになっています。一般には計算しながらプロットするものが多いので、両方を並列に操作しながらプロットするものが多いようです。

【３】　変換サブルーチン

いままで述べた各座標から他の座標への変換のうち、必要かつ十分なものがマシン語サブルーチンとして用意されています。それらは、

①　テキスト中のスクリーン・ワールド座標系→基本座標系

②　スクリーン座標系←→ワールド座標系（ＭＡＰ関数）

③　基本座標系→アドレス・ビット位置座標系

④　プレーン・アドレス・ビット位置座標系←→プレーン・バイト数・ビット位置座標系

の４つです。

①　テキスト中のスクリーン・ワールド座標系→基本座標系

BASIC文が実行されるときには、まず先頭のキーワード（中間コード化されている）によって、それぞれの処理ルーチンに分岐します。グラフィクス関係処理ルーチンは、ほとんどがROM５にあるので、それから先の構文解析・解釈実行はすべてＲＯＭ５の上で行われます。そのときにキーワードの次にある座標パラメータを得るのがこのサブルーチンです。

①－ａ　　始点座標を得るサブルーチン（始点座標ＧＥＴ：６Ｅ２ＢＨ）

これは特に１番目の座標をテキスト中から得るものです。２番目と異なることをするのは、はじめの「@」を読みとばすことと、はじめから「－」であったときにはＬＲＰ(Last Referenced Point：最後に参照した点）の座標を省略時解釈として始点とすることです。（６Ｅ２ＢＨ～６Ｅ３ＣＨ）

①－ｂ　　座標を得るサブルーチン（座標ＧＥＴ：６Ｅ３ＤＨ）

テキストから座標を得るサブルーチンです。スクリーン・ワールド両座標系を対象としていますが、その意味はWINDOW文によってウィンドウが設定されていない(ウィンドウ設定フラグ：Ｆ０８ＦＨの内容で判定できる）ときは別のルーチン（７ＤＡ７Ｈ）に飛んで座標の取り込みを行うということです。スクリーン座標に対する座標の取り込みは、①－ａでも同様に処理されています（①－ｂにはウィンドウ設定フラグを読み込んだ後にフック：ＥＥ０８Ｈへ入る処理があります)。この理由から、WINDOW文実行前では座標はすべてスクリーン座標として処理されます。

①−ｃ　　始点スクリーン座標を得るサブルーチン（７Ｄ９８Ｈ）

機能としてはスクリーン座標が対象となっただけで（すなわち得る値が整数となる）①−ａと大きな差はありません。構造もまったく同様です(フックはＥＥ０５Ｈとなります)。

①−ｄ　　スクリーン座標を得るサブルーチン（７ＤＡ７Ｈ）

①−ｃに同じ。

②　スクリーン座標系←→ワールド座標系（ＭＡＰ関数）

これはＭＡＰ関数処理の一部で、サブルーチンコールができる形になっているものです。

②−ａ　スクリーン座標→ワールド座標（６ＤＡ３Ｈ）

ビューポートとウィンドウが設定されると、その縦横それぞれの比が計算され、作業域に保存されます。ワールド座標とスクリーン座標の対応は前述の通りなので、この方程式を逆に解くことによってワールド座標を得ることができます。この変換を実行するのがこのサブルーチンです。ただし、スクリーン座標とワールド座標は（単精度実数型と整数型の表現できる数を考えれば）明らかに１対１に対応していないので、ワールド座標→スクリーン座標→ワールド座標と変換しても、始めと終わりの座標が一致するとは限りません。

②−ｂ　ワールド座標→スクリーン座標（６Ｄ８ＢＨ）

この変換は前に与えた対応の方程式を計算することによって容易に得られます。

②−ｃ　スクリーン座標←→ワールド座標エントリ（ＭＡＰ関数：６Ｄ４８Ｈ）

以上の２つのサブルーチンでは縦横どちらの座標を変換するかはそれぞれの作業域のアドレスを指定しなければなりません。しかしこのサブルーチンでは、機能コードによってそれを指定することができます（Ａレジスタで指定。ビット０が縦横：０でＸ／１でＹ、ビット１が変換の方向：０でワールド→スクリーン／１でスクリーン→ワールド。Ｆ０ＡＣＨ〜Ｆ０ＡＤＨに変換する座標値)。

③　基本座標系→アドレス・ビット位置座標系（６５１８Ｈ）

描画のときに使われる基本座標から実際にプロットするときに使われるアドレス・ビット位置座標（400ラインのときはアドレス・ビット位置・プレーン座標）への変換を行うものです。

④　プレーン・アドレス・ビット位置座標系←→プレーン・バイト数・ビット位置座標系

アドレスから先頭からのバイト数は、アドレスの上位２ビットをマスクするという単純な操作によって得られます（逆にアドレスを得るには上位２ビットを１にすればよい)。したがって、より記憶効率のよい後者の形で保存することが多くなるのは当然のことでしょう。

④－ａ　プレーン・アドレス・ビット位置座標
　　　　　　　　　　　　→プレーン・バイト数・ビット位置座標（６５Ｆ２Ｈ）

作業域のプレーン・アドレス・ビット位置情報を取り出し、プレーン・バイト数・ビット位置座標を生成するものです。

④－ｂ　プレーン・バイト数・ビット位置座標
　　　　　　　　　　　　→プレーン・アドレス・ビット位置座標（６６０６Ｈ）

ＨＬレジスタにもったプレーン・バイト数情報からアドレスとプレーンポート番号を生成するものです。200ラインモードに対してはプレーン番号はプレーン０に設定されます。

【４】　範囲のチェック

上述のような複雑な座標系をとると、画面に表示されるかされないかのチェックを行って、そのたびにどの部分が描画されるか調べなければなりません。調べ方は非常に基本的で、数を比較してフラグをセットするだけのものです。基本座標とプレーン・アドレス・ビット位置座標のそれぞれに対して用意されています。スクリーン座標とワールド座標に対してはテキストから読み込んだ時点で基本座標に変換されるため、用意されておりません。

７－１－２　描画コマンド処理の概要

他のコマンドの処理と同様に、グラフィクスのコマンドもパラメータ先頭のテキストポインタをもって処理ルーチンにジャンプしてきます。そこからは各々の処理ルーチンですべてが処理されます。よく使われる座標の取り込み処理などは、上述のようなサブルーチンになっています。他のパラメータの取り込みは、ＲＳＴ１０Ｈや数式の評価ルーチン（１１Ｄ３Ｈ）などを用い、また、『,』のチェックなどはＲＳＴ８Ｈがよく使われています。省略時の解釈についても、それぞれの場合について、メインルーチンの中で行われています。

このＢＡＳＩＣのグラフィクス関係のコマンドでは、処理内容ごとにパラメータの取り込みと実際の描画処理が比較的分離されており、実行部だけでも、パラメータがセットされていれば容易にコールすることができます。これによってユーザに対してもグラフィクス処理が行いやすくなっています。ただし、メインルーチンはたくさんの細かい領域に分かれています。

次に、その典型的な例としてLINE文のプログラムを解説することにします（７Ｅ８ＢＨ～、ＰＣ－８８０１mkIISRにおいてはＥＲＯＭ０上のもの)。

始めに座標を得ます。始点座標ＧＥＴルーチンによって始点座標、座標ＧＥＴルーチンによって終点座標を得ます。２つの座標を結ぶマイナス記号『－』は「ＲＳＴ８Ｈ　ＤＢ　“－”」でチェックします。ＲＳＴ８Ｈは、次の文字がこの命令の次にある文字と同じかどうか比較し

て異なっていれば「Syntax error」とし、同時に次の１文字を得るものです。

そして、カラーパレット番号を得るサブルーチンをコールします。ここでは、省略時解釈についてどのように処理されるかを見ることができます。まず、省略時の解釈（＝フォアグラウンドカラー番号：Ｆ０１ＥＨ）をレジスタに代入します。それから、ＲＳＴ１０Ｈによって次の１文字を取り出し、文末だった場合（Ｚフラグ＝"１"）は省略時解釈をセットして終了します。そうでない場合には、コンマ『,』でければならないので、ＲＳＴ８Ｈでそのことをチェックし、その次の文字も『,』であるときにはパラメータが省略されているので文末の場合と同様に処理します。獲得した値は、色マスクパターン生成（６６Ａ０Ｈ）でプレーンごとの成分に分解されます。

次に、ラインスタイル値に省略時解釈値を代入し、すでに文末であることが検出されている場合には、そのまま直線描画メインルーチン（７ＥＦ５Ｈ）に進みます。

7－2　グラフィクス処理のアルゴリズム

7－2－1　直線の発生

【1】曲線の方程式とアルゴリズム

曲線を発生するには、まず曲線を表す方程式を求め、それを用いて実際に計算することによって曲線上の各点を求めることが基本です。このときの曲線の方程式は、描画の効率に決定的な影響を与えます。また、効率だけではなく、得られる曲線の質を左右するものでもあります。たとえば、「ＰＣ－８８０１　N_{88}－ＢＡＳＩＣグラフィクス解析マニュアル」(尾高保　著)における２つの円弧発生アルゴリズムについての考察を参照してみてください。

それでは、どのような方程式が得られれば、効率的で、しかも美しい曲線が得られるのでしょうか。

ここでは、横軸にｘ座標、縦軸にｙ座標をとり、曲線を表す方程式、

$$(1)\qquad \frac{dy}{dx} = f(x, y)$$

について考えることにします。なぜこの形を考えるかというと、後述の直線と円弧の方程式においてはこの形に変形すると単純な和や差の繰り返しだけで計算することができることと、この方法がＰＣ－８８０１にも組み込まれていることからです。

式（１）が具体的な形で与えられたとき、これを数値的に解けば求める曲線が得られます。ただし、一般には式（１）を満たす曲線は無数にあるので、さらに、条件として曲線の始点の座標を与えることにします。これによって曲線は一意に定まります。もちろん、計算中に誤差が出てきますが、目的がグラフィクスなので、誤差が１ドット未満になるようにパラメータを選べば十分です。

初期値問題（１）を数値的に解くには多くの方法がありますが、上に述べた理由から、１次のEuler法を用いることにします。

これは、最も簡単な方法の１つで、ｈを十分小さい正数とするとき、ｙをｘの関数であることを明示して、ｙ(ｘ）と書くと、微分係数の定義から、

$$y(x+h)-y(x) \fallingdotseq \frac{dy}{dx}\cdot h$$

となることを利用して ｙ(ｘ) を近似するものです。したがって、簡単にするために初期条件を「ｙ(０)＝ａ」とすると、

$$(2)\qquad y(m\cdot h) = y((m-1)\cdot h)+h\cdot f((m-1)\cdot h,\ y((m-1)\cdot h))$$

（ただしｍは正の整数）

と帰納的に求めることができます。もちろん、ｍが大きくなるにつれて誤差が累積するので、ｈとｍは十分小さくなければなりません。逆に、ｈとｍを十分小さくとれば、精度はそれなりに保証されます。

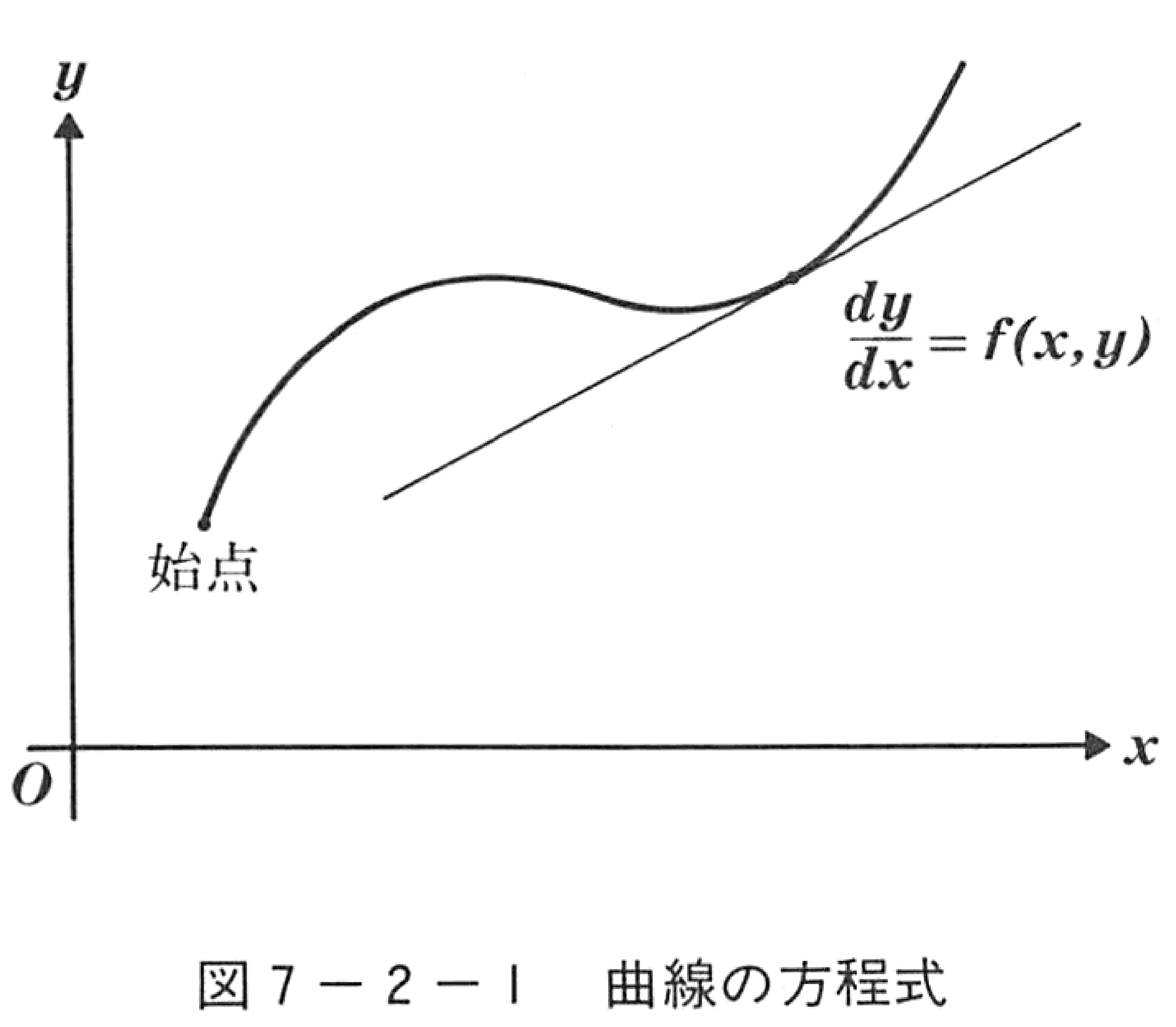


図７－２－１　曲線の方程式

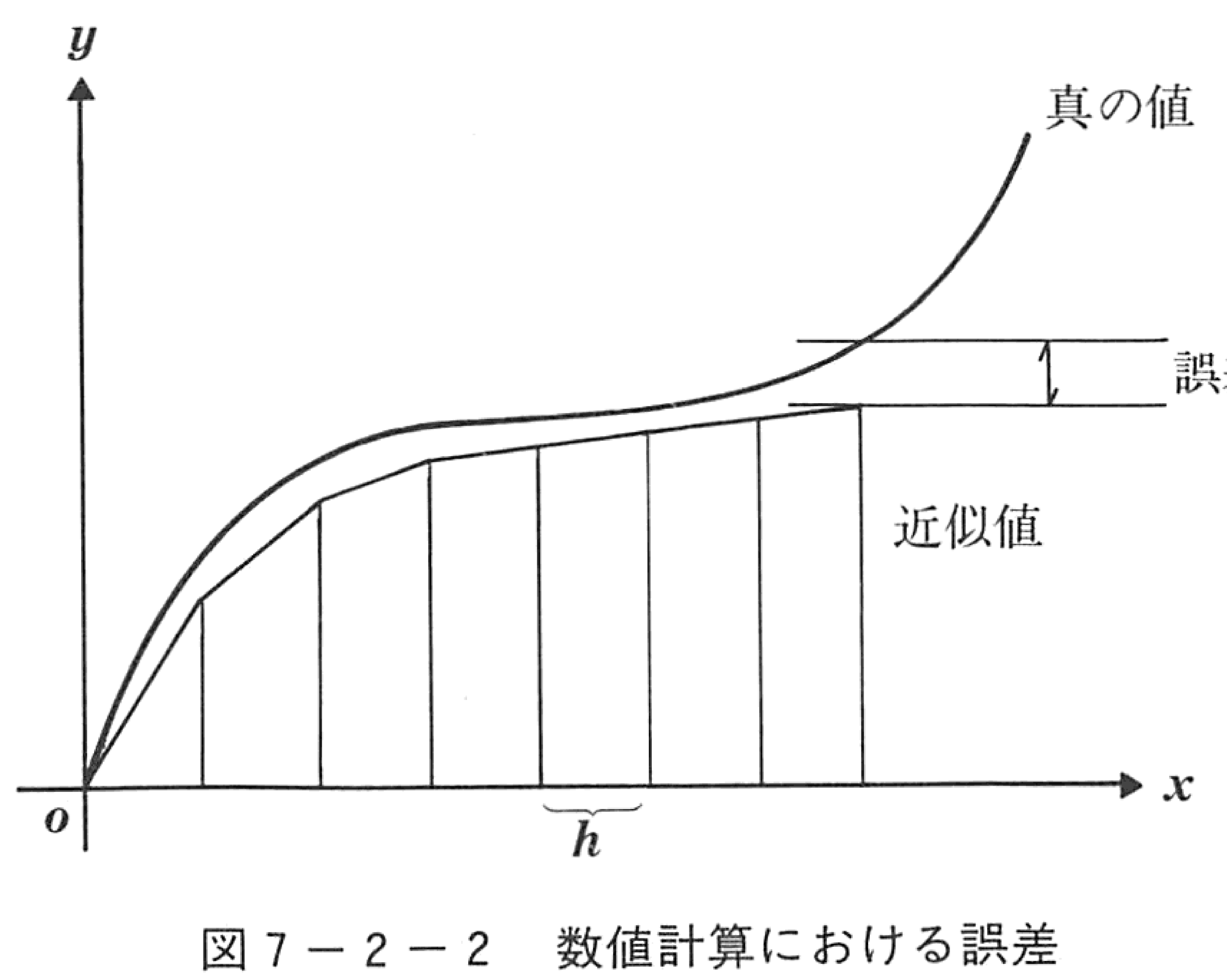


図７－２－２　数値計算における誤差

これまで述べたことから、実際に曲線を計算するアルゴリズムは、曲線の方程式が（1）で与えられ、（x，y）は（0，a）を通る求める曲線上の点、hは十分小さい正数とすると、次のように書くことができます。

① x←1，y←a
② （x，y）は新しい点である
（このときプロットすれば、画面上に曲線が得られる）
③ y←y＋h・f（x，y），x←x＋h
④ 終わりでなければ、②へ戻る

ちなみに、この方法を一般化したものは、DDA（Digital Differencial Analyzer）と呼ばれます。DDAにおいては、まず、問題を連立微分方程式で記述し（上記の場合はこのままでよい）、それに従って上述の積分器（実質的に積分を行っている）を組み合わせ（これをマッピングという）、次に各積分器のレジスタの数値表現（ビット数、小数点の位置、負数の表現形式など）を定め（これをスケーリングという）、それによって微分方程式を解くことができます。一口にDDAといっても、積分器に、1次のEuler法（＝矩形公式）を用いているとは限らず、台形公式などを用いる場合もあります。

以上は微分方程式（1）を数値的に解くということについてでしたが、グラフィクス用曲線発生に用いる場合には、さらに各数値の整数部だけしか用いられないということに注意しなければなりません。したがって、一般の場合には数値を四捨五入してプロットすることが多いようです。事実、PC－8801もそのようになっています。

【2】　直線の方程式と直線発生

グラフィクスにおいて直線という場合には、始点と終点の2点が与えられ、それらを結ぶ線分を意味します。よって、始点座標（x_0，y_0）、終点座標（x_1，y_1）に対して、この直線を表す方程式は、

$$(3)\qquad y - y_0 = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0)$$

（ただし、$x_0 \neq x_1$とする）

となります。【1】の方法を適用するためには、（1）の形で表示する必要があります。
そこで、（3）の両辺をxで微分すると、

$$(3')\qquad \frac{dy}{dx} = \frac{y_1 - y_0}{x_1 - x_0}$$

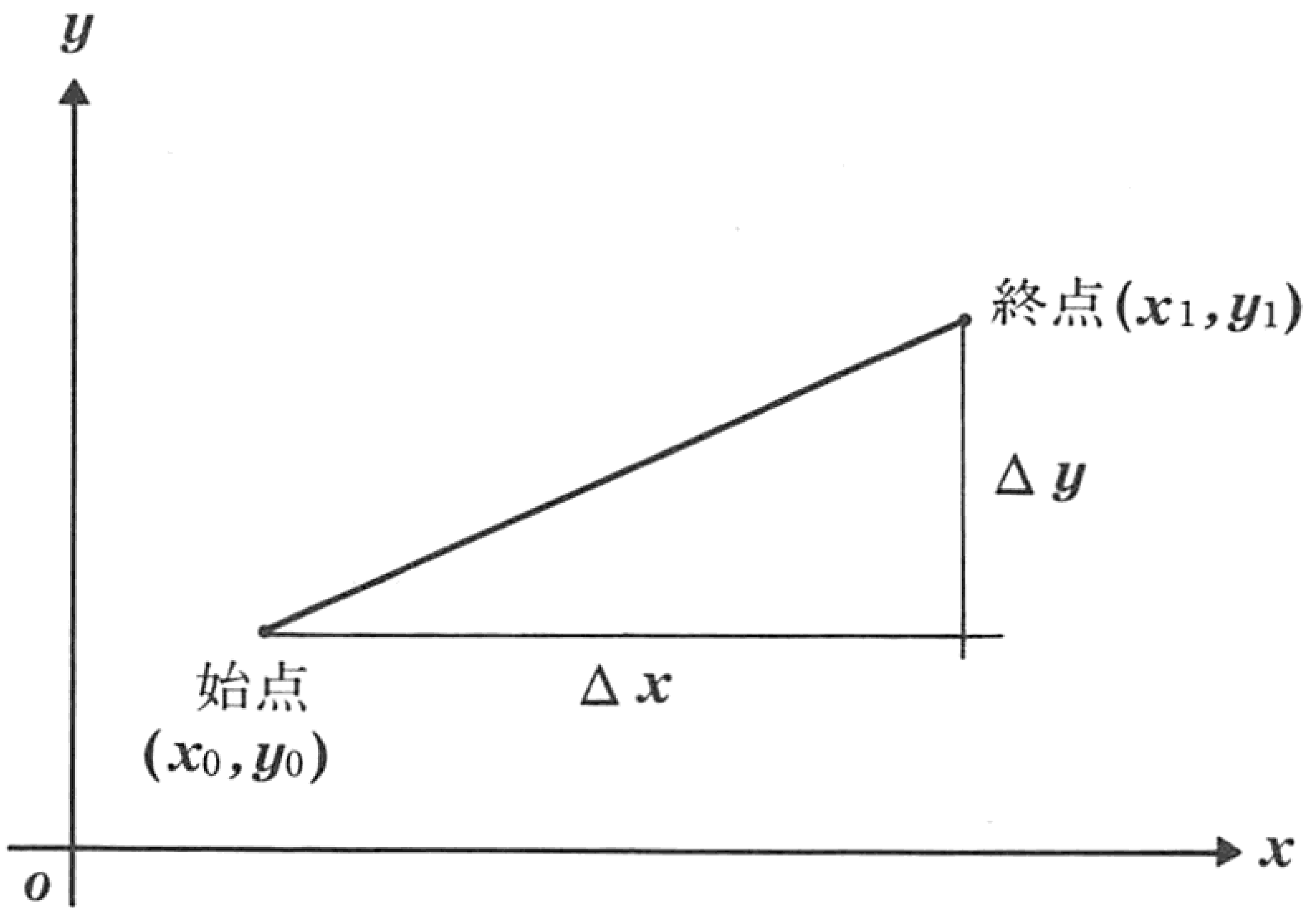


図7－2－3　グラフィクスにおける直線

また、始点が（x_0, y_0）であることから$y(x_0)=y_0$が得られます。これはちょうど（1）の形になっているので【1】を用いて計算することができます。

ところが、これはさらに効率よくすることができます。

その方法は、座標を適当に取り直すことによって、すべて始点(0, 0)：$0 \leqq y_1 - y_0 \leqq x_1 - x_0$の場合に帰着することができるということを利用します。座標を取り直すとはどういうことかというと、$\Delta x = x_1 - x_0$, $\Delta y = y_1 - y_0$とするとき、$\Delta x \leqq \Delta y$ならば、u軸をy軸と平行に、Δyの負・非負によってそれぞれ反対向き・同じ向きに定め、v軸はx軸と平行にΔxの符号によって同様に定めます。$\Delta x < \Delta y$のときは、xとyを入れ替えて考え、同様に定めます。このとき原点は始点としておきます。このように*u-v*座標をとっても、プロットするときの手間はまったく変わりません。それは、座標u, vの向きを設定するときに、その向きにあわせてプロットする点を動かすサブルーチンのアドレスを設定すればよいからです。

$0 \leqq y_1 - y_0 \leqq x_1 - x_0$とした場合に何が有利かというと、直線を描くのに必要な点の数が$(x_1 - x_0 + 1)$に等しいということです。したがって、u座標を0から$(x_1 - x_0)$まで1ずつ増加させながら計算・プロットしていけば、自然に2つ重なったり離れたりせずに描くことができます。

【1】に当てはめて、直線を得るアルゴリズムを作ってみましょう。

さきほど述べた理由から、hは1で十分であることがわかります。また、*u-v*座標についてプロットできれば基本座標でプロットできるので、*u-v*座標だけについて考えます。

また、プロットするときには、いちばん近い整数座標（vに関して）にプロットしたいので、四捨五入してv座標を求めます。すなわち、vに1/2を加えて整数部分だけを取り出すのですが、毎回行わなくてもvの初期値として1/2をとっておけば、あとは整数部分を得るだけで目的は達成されます。

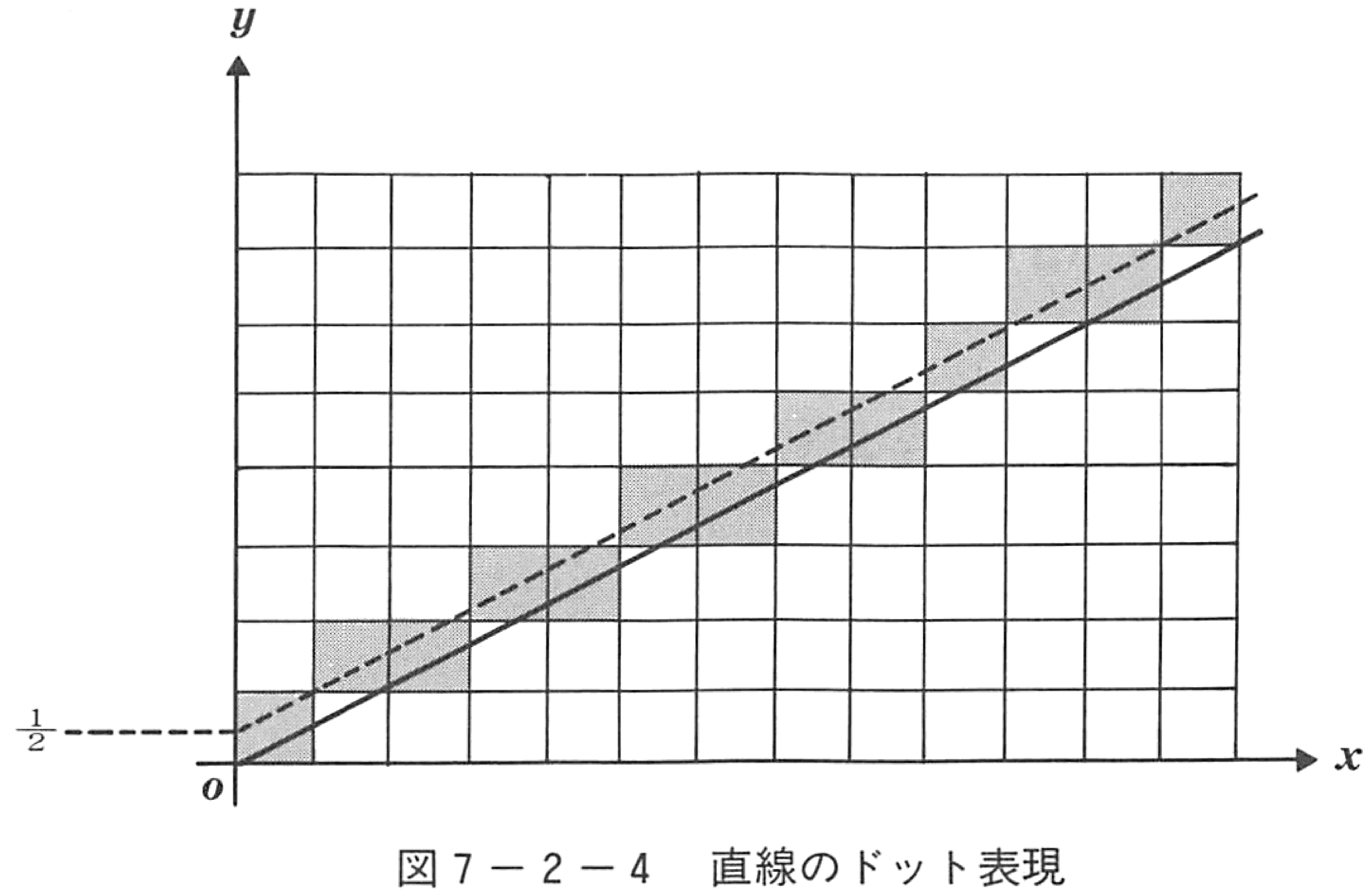


図7－2－4　直線のドット表現

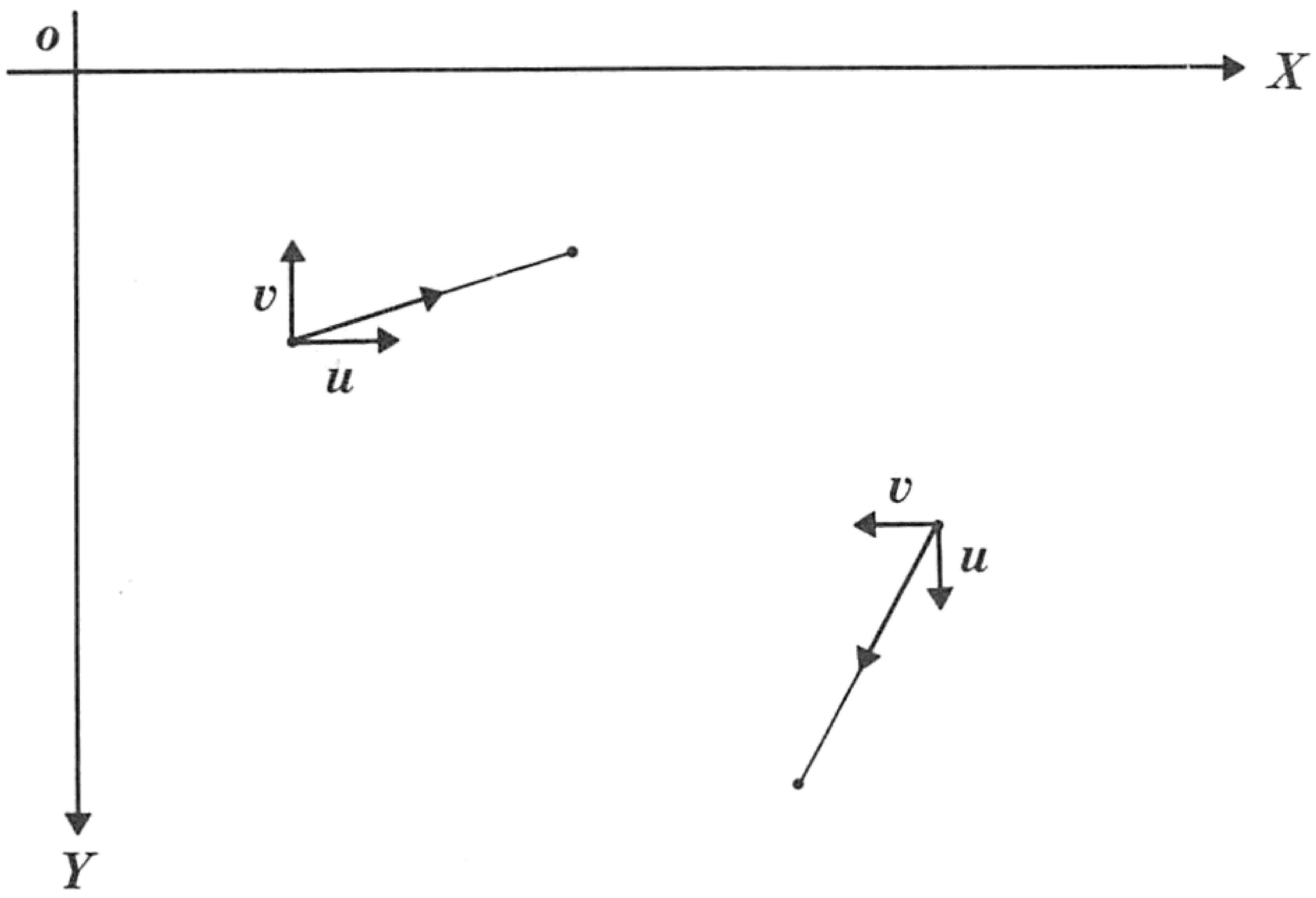


図７－２－５　座標の取り直し

これらの点を考慮すると次に示すアルゴリズムが得られます。

① u←0，v←1/2
② （u，v）をプロット（プロットするときに整数部分だけを取り出す）
③ u＝Δxなら終了
④ v←v＋Δy／Δx
⑤ u←u＋1
⑥ ②へ飛ぶ

また、Δy／Δx≦1より、整数部分を取り出す代わりに、vを始めから整数部と小数部に分けておいて小数部が1以上になったときに整数部を増加（＋1）させると同時に小数部を再び1未満の数とする、というように変更します。こうすればもう、*u-v*座標を特に意識しなくても、小数部があふれるたびにプロットする点をv方向に動かせばよいことになります。こうして次のアルゴリズムを得ます（ΔuはΔxの絶対値とΔyの絶対値の大きい方、Δvはそうでない方とする)。

① x←x_0，y←y_0，f←1/2（vの小数部），c←Δu＋1（点の数）
② （x，y）をプロット
③ c＝0なら終了
④ f←f＋Δv／Δu
⑤ f≧1 なら f←f－1，点をv方向に1ドット動かす
⑥ 点をu方向に1ドット動かす，c←c－1
⑦ ②へ飛ぶ

さて、ステップ④では除算が使われています。よく知られているように、Z80で除算を行うのは簡単なことではではありません。また、あらかじめ計算したものを使用すれば、速度の点では問題なくできますが、有限桁の2進表示では誤差が出る可能性があります。そこで、fのかわりにfをΔu倍したものを考えます。この変数をtとすると、次のようになります。

① $x \leftarrow x_0$, $y \leftarrow y_0$, $t \leftarrow \Delta u/2$, $c \leftarrow \Delta u + 1$
② (x, y)にプロット
③ c＝0なら終了
④ $t \leftarrow t + \Delta u$
⑤ $t \geqq \Delta v$ なら $t \leftarrow t - \Delta v$ として点をv方向に1ドット動かす
⑥ 点をu方向に1ドット動かす，cにc－1を代入
⑦ ②へ飛ぶ

これは、PC－8801の直線発生ルーチンと同値なことを行っています。より詳しくいえば、順番が異なっており、PC－8801での実行の手順は次のとおりです。

①'	$x \leftarrow x_0$, $y \leftarrow y_0$, $p \leftarrow -\Delta v$, $t \leftarrow \Delta u$, $c \leftarrow \Delta u + 1$	(①に相当)
②'	c＝0なら終了	(③)
③'	u方向へ1ドット動かす	(⑥の前半)
④'	(x, y)へプロット	(②)
⑤'	$c \leftarrow c - 1$	(⑥の後半)
⑥'	$q \leftarrow t + \Delta u$, $r \leftarrow q + p$	(④)
⑦'	$q \geqq 0$ なら $t \leftarrow r$, ②'へ飛ぶ	(⑤の条件)
⑧'	$t \leftarrow q$	(⑤の処理前半)
⑨'	c＝0なら終了	(③)
⑩'	v方向へ1ドット動かす	(⑤の処理後半)
⑪'	③'へ飛ぶ	

【3】　実際のＬＩＮＥ文

LINE文には、単に直線を引くだけでなく、長方形を描いたり、内部が塗られている長方形を描くなどの機能もあります。また、直線を引くときでも、ラインスタイルを指定することができます。一方、【2】では始点と終点が画面内にある場合だけを想定しましたが、LINE文ではワールド座標も扱うので、そうでないときの処理も考えなければなりません。同様にビューポート内にあるかどうかの処理も行い、ビューポート外の部分は描かないようにする必要があります。

長方形を描いたり内部を塗ることは比較的単純なので省略し、後半のラインスタイルや表示されない部分の処理（クリッピングという）を含めたメインルーチンについて述べることにします。

直線描画処理（７Ｆ４７Ｈ〜７ＦＤ４Ｈ）は、

① ビューポート外の部分のクリッピング

② 増分の計算と点移動サブルーチンの設定

③ 直線上の点の計算

④ ラインスタイルに合わせたプロット

に分けることができます。

ビューポート外の部分のクリッピング（７Ｆ４ＡＨ〜７Ｆ４ＤＨ，サブルーチン：６Ｆ７ＡＨ〜７０５２Ｈ）の考え方は、比例関係を用いるものです。

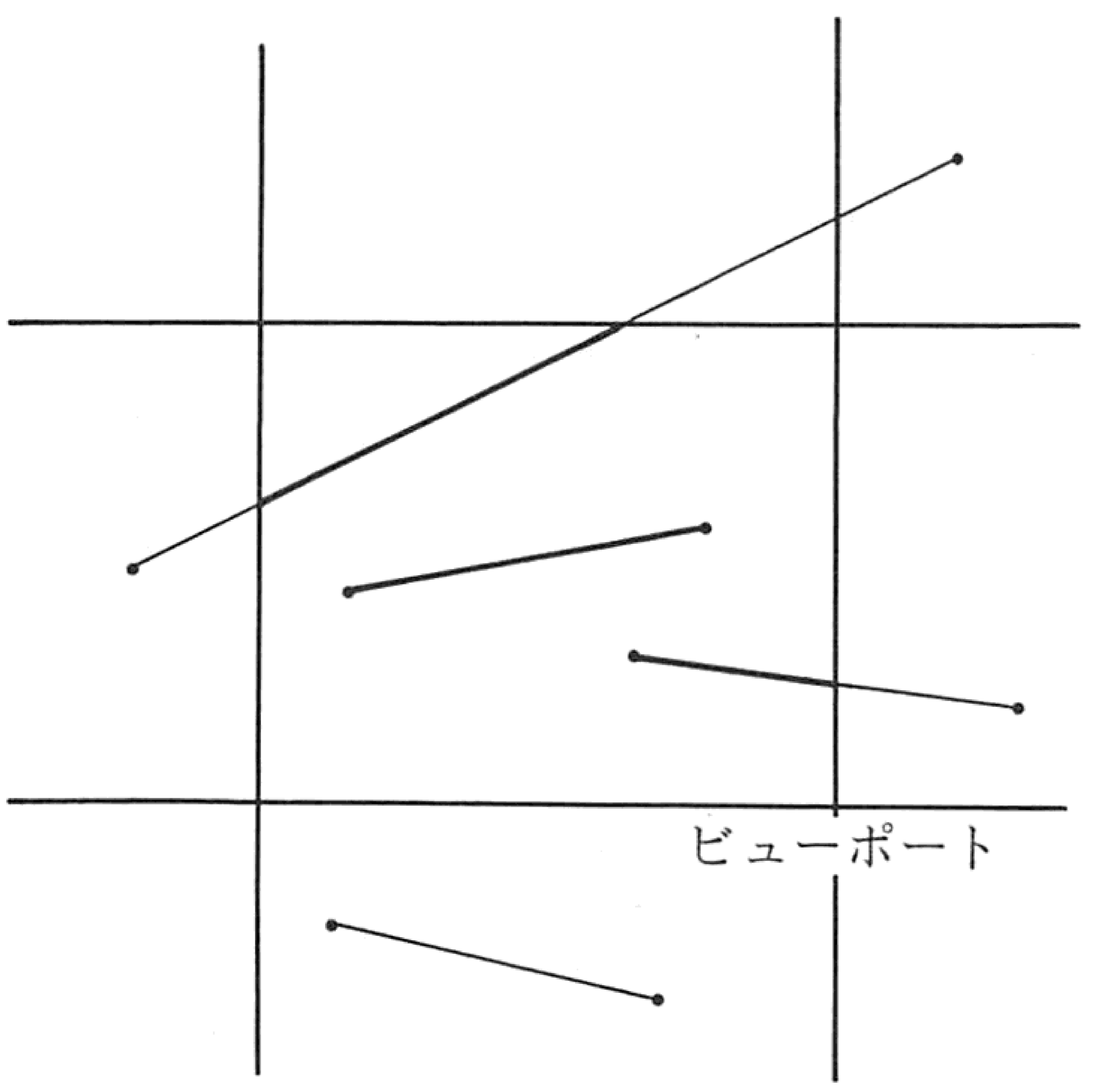


図７−２−６　クリッピング

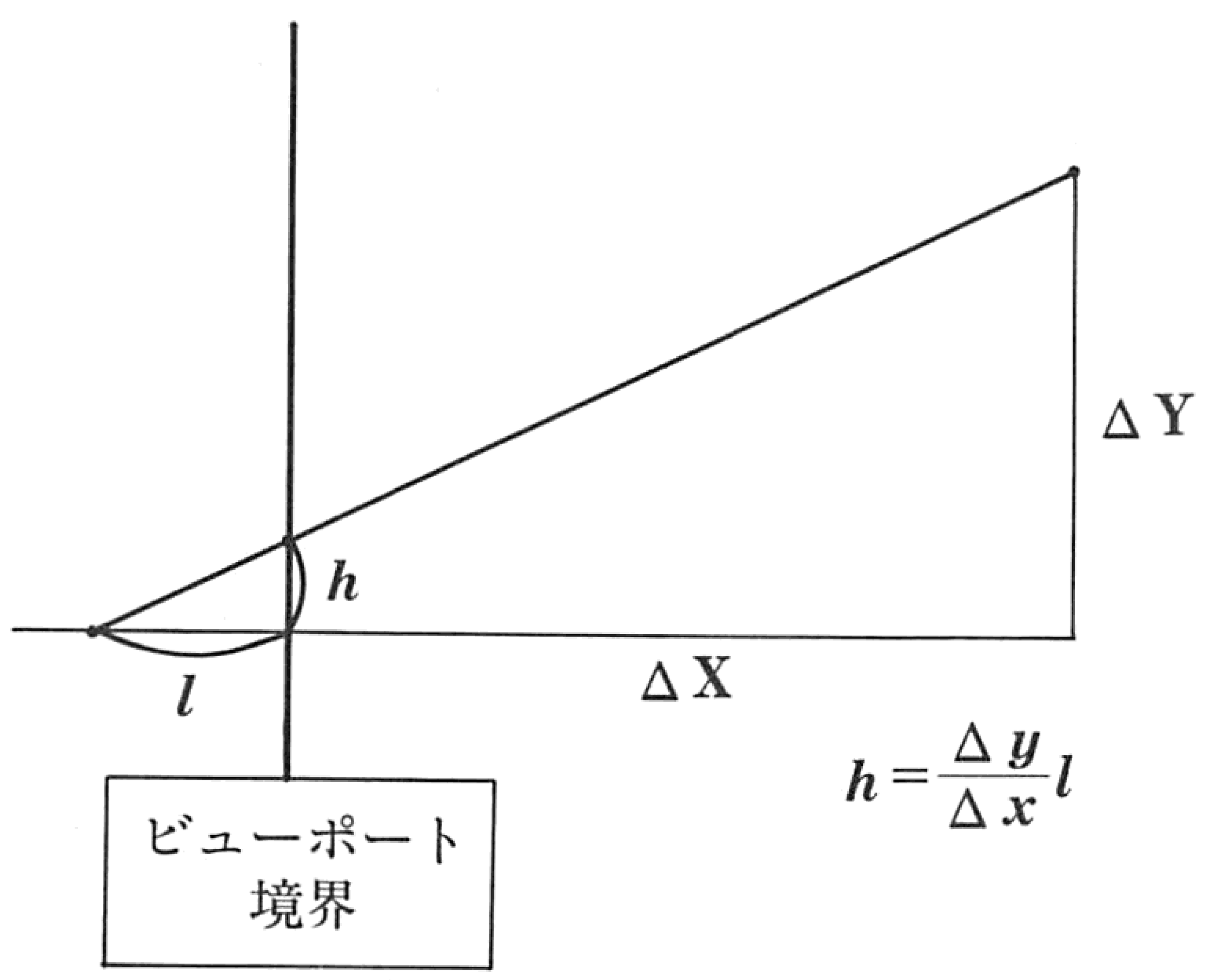


図７−２−７　交点の計算

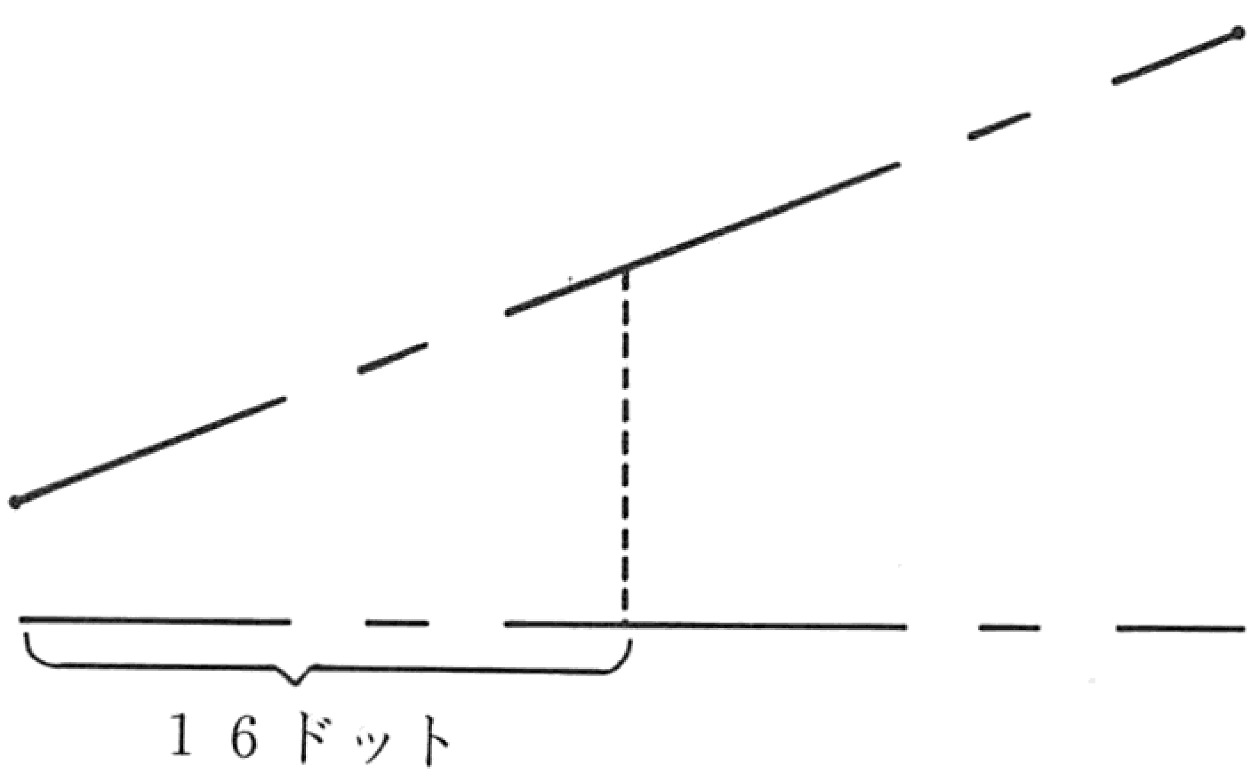


図７−２−８　ラインスタイル

いま、Ｘについて考えることにします。始点から終点のＸ，Ｙの増分をΔＸ，ΔＹとすると、始点からｘだけＸ座標から増えるとＹ座標は(ΔＹ／ΔＸ)・ｘだけ増加します。したがって、ビューポートの（たとえば基本座標形における）Ｘの左端の座標をV_1とすると、(V_1−Ｘ）ΔＹ／ΔＸ＋Ｙがビューポート左端と直線との交点のＹ座標となります。これをさらにビューポート

の右端の座標V_rについて行えば、X座標についてビューポートの範囲内に収めることができます。また、このX，Yを入れ替え、ビューポートの上端下端の座標V_h，V_dについて同じサブルーチンを呼べば、直線のビューポート範囲内での始点と終点を得ることができます。

ただし、これは直線の端点がビューポート外にはみ出しているときの話で、そうでないときは無意味です。それを調べるためのサブルーチン（６Ｆ７ＡＨ）が用意されています。クリッピング処理（６Ｆ９２Ｈ）では、まずこのサブルーチンを呼んでから、はみ出している点についてクリッピングを行います。X，Yどちらの座標で範囲からはみ出しているかはわかっている（６Ｆ７ＡＨによってＨ：終点、Ｌ：始点に返される）のでそれに応じて、Xに対するクリッピング（６ＦＤ６Ｈ）、Yに対するクリッピング（７０３０Ｈ）を呼びます。その後、さらにまだはみ出している場合があるのでこれを繰り返します。

増分の計算と点移動サブルーチンの設定（７Ｆ４ＥＨ～７Ｆ８７Ｈ）は【２】の座標の取り直しに相当します。ただし、【２】では触れませんでしたが、Yの増分が負の場合は始点と終点を交換することでYに対して動く方向は常に正の方向とすることができます。こうすると、場合分けによって設定するサブルーチンが少なくなり、場合分けに要する手間も半減することができるという利点があります。なお、このときの点移動サブルーチンはアドレス・ビット位置座標系に対するものです。続いてXの増分の符号によってX方向に動かすサブルーチンアドレスを選択します。【２】に合わせていえば、u方向のサブルーチンアドレスはＥ７Ｅ６Ｈ～Ｅ７Ｅ７Ｈ、v方向はＥ７Ｅ３Ｈ～Ｅ７Ｅ４Ｈにセットされています。Ｅ７Ｅ２Ｈ～Ｅ７Ｅ５Ｈには、ＪＰ命令のＣ３Ｈが入っており、それぞれのアドレスをコールすると、ちょうど点移動サブルーチンがコールされることになります。サブルーチンアドレスの設定と同時に、v方向に取られた座標についての増分を作業域（Ｆ０２３Ｈ～Ｆ０２４Ｈ）にストアします。u方向の増分の絶対値はＨＬレジスタに保持され、次の処理に渡されます。

直線上の点の計算（７Ｆ８８Ｈ～７ＦＢＣＨ）はほぼ【２】に示したとおりです。ただし、座標に関しては１ドット移動し、プロットするには基本座標が用いられている、ということは先に述べました。また、前処理からわたされたu方向の増分の絶対値は、符号を反転させて作業域（Ｆ０２１Ｈ～Ｆ０２２Ｈ）にストアされ、比較の際に用いられます。ちなみに、v方向の増分も作業域（Ｆ０２３Ｈ～Ｆ０２４Ｈ）にストアされています。より詳しい情報については、「グラフィクス解析マニュアル」（尾高　著）もしくは、ＲＯＭを直接に参照してください。

ラインスタイルに合わせたプロット（７ＦＣ５Ｈ～７ＦＤ４Ｈ）は、ラインスタイルバッファ（Ｆ０２５Ｈ～Ｆ０２６Ｈ）に設定されている内容によってプロットを行います。プロットするときの条件はコールしたときにバッファの最上位ビットが１であることです。ラインスタイルは16ビット左回転されます。計算のアルゴリズムから同じ点に２回はプロットしないので、画面にはラインスタイルのパターンがちょうど増分の大きい方の16ドットに１パターンの割合で張り付けられたように表示されます。

7－2－2　円弧の発生

【1】　円弧発生のアルゴリズム

直線の場合と同じように、円の方程式を考え、そこから円周上の点を計算するアルゴリズムを求めてみましょう。

まず円の方程式ですが、仮想座標系u-vを設定し、中心（0，0）、半径aの円を考えます。この円を表す方程式は、

（4）　$u^2 + v^2 = a^2$

となります。これを（1）の形で表すためにuで微分し、整理すると、

（4'）　$\dfrac{dv}{du} = -\dfrac{u}{v}$

また、円を計算し始める点を（0，a）とすると、v(0)＝aを得ます。逆に、(4')とv(0)＝aという2つの条件を満たすものは(4)に限ります。

さきほどの7－2－1【1】に合わせて得られるアルゴリズムを書くと、

① u←0，v←a，h←1/2，v←v－1/2
② （u，v）をプロット（ただし、u，vの四捨五入した値を用いる）
③ u≧vなら終了
④ v←v－（u/v）・h
⑤ u←u＋h
⑥ ②へ戻る

②で四捨五入する理由は直線の場合と同じです。また、h＝1/2はグラフィクスに関しては十分な値です(現に、PC－8801ではこの値が用いられています)。さらに、①において、やはり直線と同様な理由で1/2小さく初期値を設定します。

③は停止するための条件ですが、これでは円の8分の1しか得られないということになります。それにもかかわらずなぜこのようになっているかというと、vが0に近づくにつれて微分係数が負の無限大に近づくので、そのために誤差が少なくすませるためにはuをあまり大きくさせられないからです。円全体を得るにはこの点を$\pi/2$，π，$3\pi/2$だけ回転させたものと、直線u＝vに対して対称なものをやはり同じように回転させた点すべてを求めれば得られます。しかし、現段階ではプロットするときのことはあまり深く考えずに、円の8分の1の点が得られるということだけに注意してください。というのは実際にCIRCLE文がプロットするときには楕円である場合を考慮しなければならないからです。

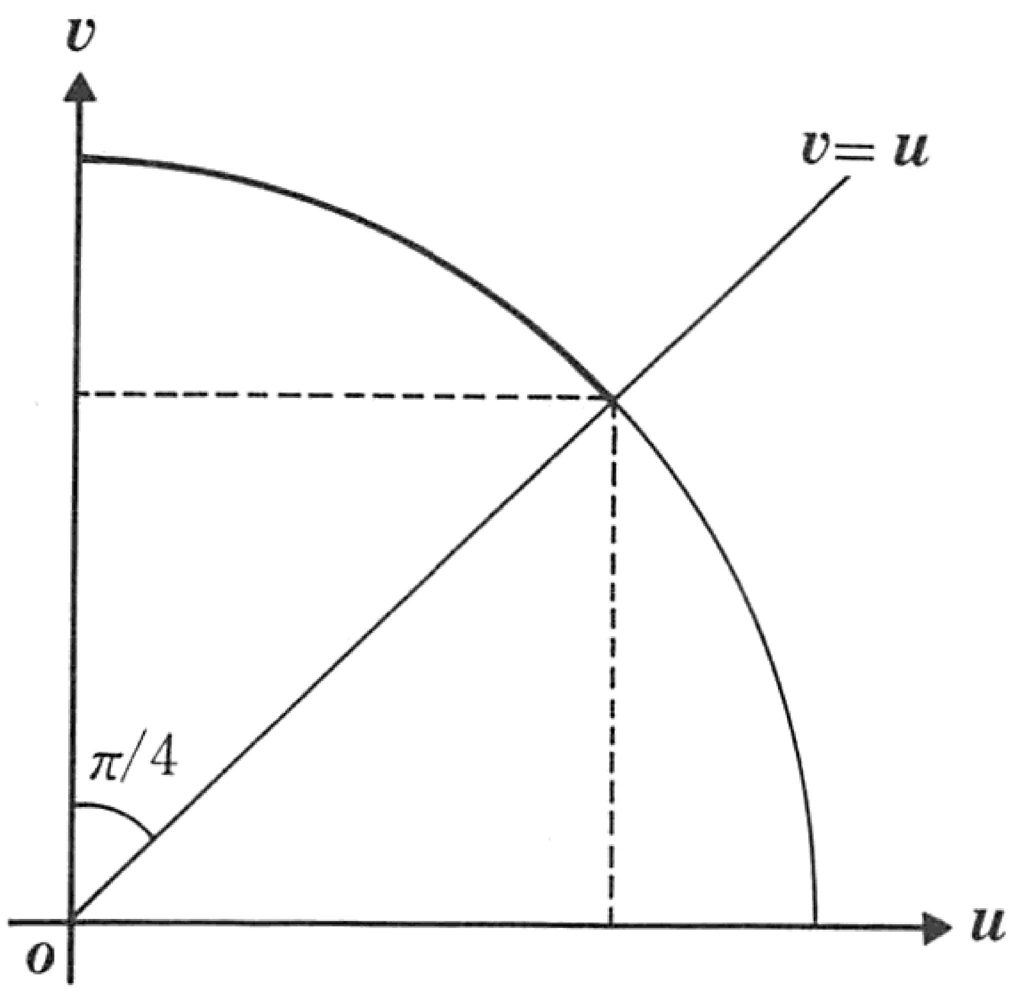


図7－2－9　計算する円弧

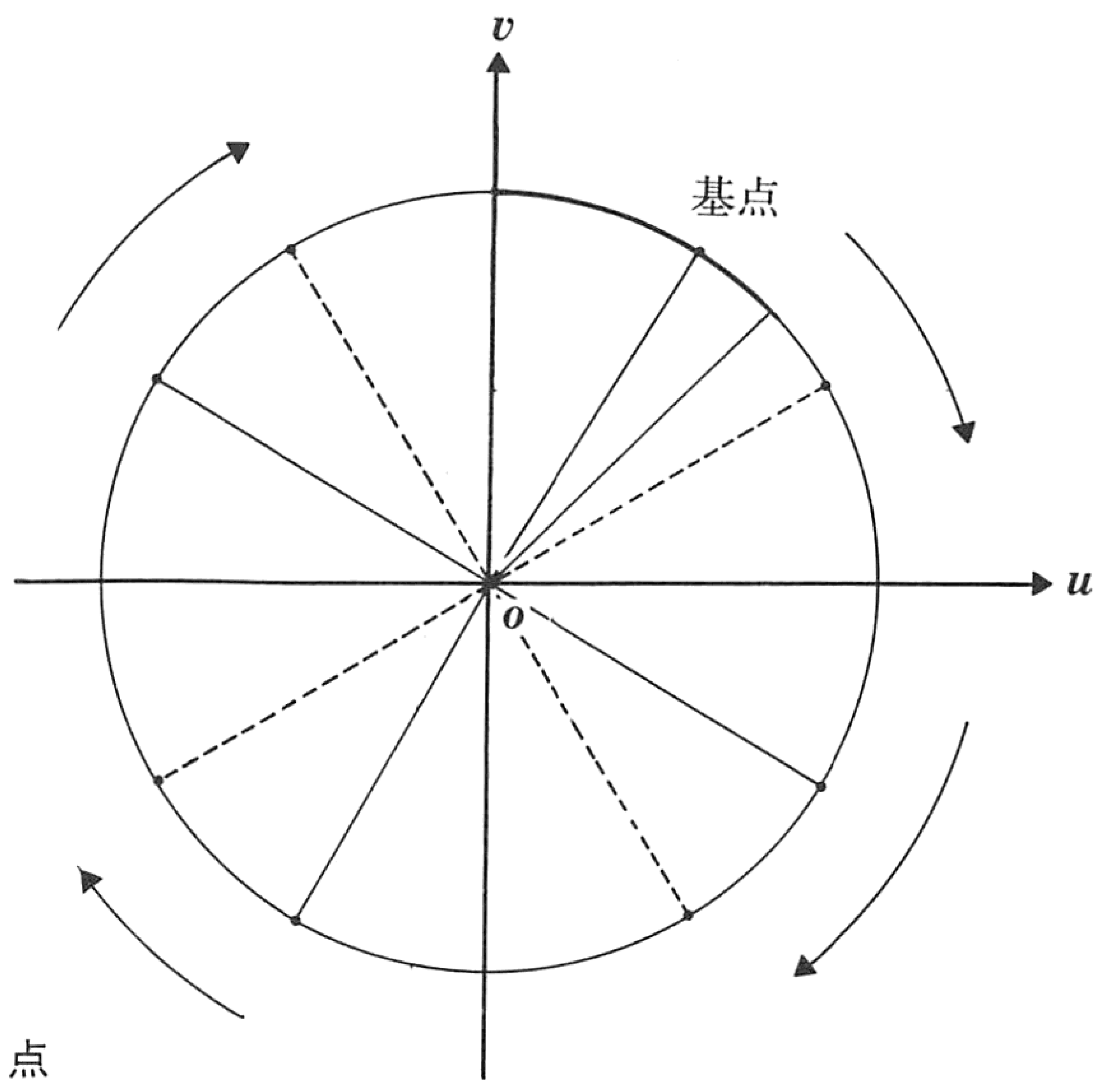


図7－2－10　基点と、その回転による7点

直線の場合にならって、このアルゴリズムをさらに効率のよいものに変形します。

まず、vを1/2以上の部分（あらためてvとおく）と1/2未満の部分fに分解し、fがあふれた（1/2以上になるとき）ならば、それにしたがってvを変化させるようにします。

① u←0，v←a，h←1/2，f←0，v←v－1/2
② （u，v）をプロット（前と同様に四捨五入する）
③ u≧vなら終了
④ f←f＋（u／v）・h
⑤ f≧h なら f←f－h，v←v－h
⑥ u←u＋h
⑦ ②へ戻る

④ではf≪vであることから、u／v≒u／（v＋f）であることを利用しています。

さらに、fの1/h倍をtとし、④の除算を取り除くと、

① u←0，v←a，h←1/2，t←0，v←v－1/2
② （u，v）をプロット（四捨五入する、h＝1/2だから）
③ u≧vなら終了
④ t←t＋u
⑤ t≧v なら t←t－v，v←v－h
⑥ u←u＋h
⑦ ②へ戻る

もちろん④では、u／v ≦１ であることから得られる座標は１ドットより多く離れてしまうことはありません。直線の場合と違って④⑤における増分値に相当するものは変化しますが、それでも計算上なんの問題もないことに注意してください。

ここから、ＰＣ－８８０１で用いられているアルゴリズムに変形するのは容易です。改良の主な点は余分な点をプロットしないようにすること、始めに行っていたｖの1/2補正をアルゴリズムの中に組み込んだこと、ｔとｖの大小を毎回比較しなくても済むようにしたことです（厳密には等価とはいえませんが、ほとんど変わりません）。

① u'←０，ｖ'←２ａ，ｔ←０
② u'が奇数なら⑤へ進む
③ ｕ←u'/2を四捨五入，ｖ←v'/2を四捨五入
④ （ｕ，ｖ）をプロット（実際には８点プロット）
⑤ u'≧v'なら終了
⑥ ｔ←ｔ＋u'＋１
⑦ ｔ＜０なら⑨へ進む
⑧ ｔ←ｔ－v'＋１，v'←v'－１
⑨ u'←u'＋１
⑩ ②へ戻る

u'，v'は、それぞれｕ，ｖを２倍したものです。②において余分な点（1/2ドット分ずつｕが増えるので２回に１回はプロットしない方がよい点となる）をプロットしないようにしています。⑥⑧における＋１は誤差補正のためと思われます。１回目に⑥⑦⑧と通るので、⑦のように正負の判定だけで済ませることができます（本来このような見方でなく、u'，v'は16ビット表現で小数点が第１ビットと第０ビットの間にあるというように解決すべきかもしれませんが、プログラムとの比較を容易にするためにこのような表現をとりました）。

【２】 実際のＣＩＲＣＬＥ文

CIRCLE文はLINE文の場合とは異なり、多くのパラメータ指定をすることができます。ここではまず、これらのパラメータはどのような意味を持ち、どのような形で保存されるかを解説します。

まず、中心のワールド座標と半径を得ます（７９Ｄ６Ｈ～７９Ｆ０Ｈ）。これは意味を説明するまでもないでしょう。ワールド座標系で指定されますが、他と同様に基本座標系に変換されます。半径についてはX方向についての長さに変換されます。さらにこの値を用いて全周（仮想座標における円を考えている）の８分の１のドット数を計算します（７９Ｆ１Ｈ～７Ａ０８Ｈ）。どのように計算するかは、このアルゴリズムでは毎回ｕが増加し、プロットする点の数は

ｕ方向に動いた数と一致します。したがって、（半径）× sin（$\pi/4$）と近似的に計算することができます。誤差はあまり問題となりません。八分円のドット数は作業域（Ｆ０７ＣＨ～Ｆ０７ＤＨ）に保持されます。

次に、カラーパレット番号を得た後、円弧の開始・終了角度を得ます（７Ａ１３Ｈ～７Ａ３９Ｈ）。この処理の主要部はサブルーチン化されており（７ＢＤ７Ｈ～７Ｃ３２Ｈ）、始点・終点ともに共通に用いています。処理内容は、角度の値が負であるときは中心へ直線を引くフラグ（Ｆ０７ＢＨ）をセットします（第０ビットが小さい方の角度、第７ビットが大きい方の角度）。値は大きさだけが角度としての意味を持ち、単位はラジアンです。この値からその点が基準点、すなわち、仮想座標における（０，半径）から何ドット目であるかを計算します（単純に全周ドット数×角度/２πを計算すればよい）。そして、開始・終了のうち小さい方、大きい方をそれぞれ作業域（Ｆ０７４Ｈ～Ｆ０７５Ｈ，Ｆ０７６Ｈ～Ｆ０７７Ｈ）にストアし、どちらが開始であるかを開始角度フラグ（Ｆ０７ＡＨ：０で小さい方が開始角度）に保持しておきます。

そして、最後のパラメータとして楕円率を得ます（７Ａ３９Ｈ～７Ａ６８Ｈ）。楕円率はＸ方向とＹ方向の半径の比ですが、プロットに際して点と点の間にすき間があかないようにするために、必ず本来の円より内側にはいるように値が解釈されます。すなわち楕円率は、１以下のときにはＹ方向に乗ぜられ、１より大きいときにはその逆数がＸ方向に乗ぜられます。したがって、保持されるのは、Ｘ，Ｙのどちらの方向に楕円率（またはその逆数）が乗ぜられるかの楕円率フラグ（Ｆ０８４Ｈ：楕円率が１より大きいとき１）、楕円率（またはその逆数）は256倍したものが整数化されて作業域（Ｆ０７２Ｈ～Ｆ０７３Ｈ）にストアされます。

パラメータをすべて獲得した後、メインルーチン（７Ａ６９Ｈ～７ＡＢ１Ｈ）を実行します。この内容はすでに述べたとおりです。ｕ－ｖ座標の点を８点プロットルーチンに渡し、全体の円を得ます。このルーチンについてのさらに詳しい情報は「グラフィクス解析マニュアル」（尾高　著）またはＲＯＭを直接に参照してください。

８点プロットルーチン（７ＡＢＥＨ～７Ｂ８６Ｈ）の処理の内容は少し複雑で、*u-v*座標の点（ｕ，ｖ）とそれを$\pi/2$，π，３$\pi/2$だけ時計回りに回転させたものに対する基本座標を求め（こ

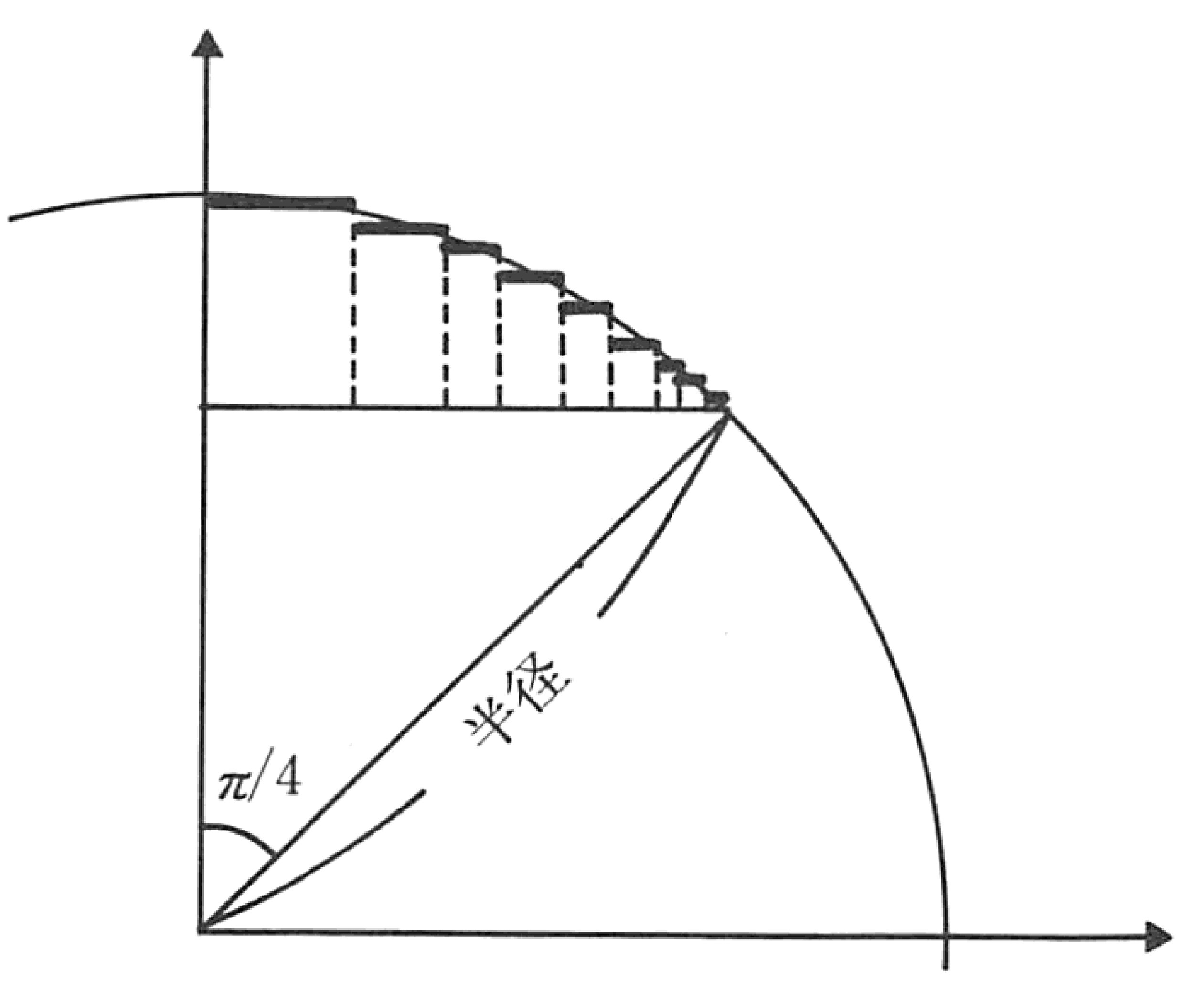


図７－２－11　八分円のドット数

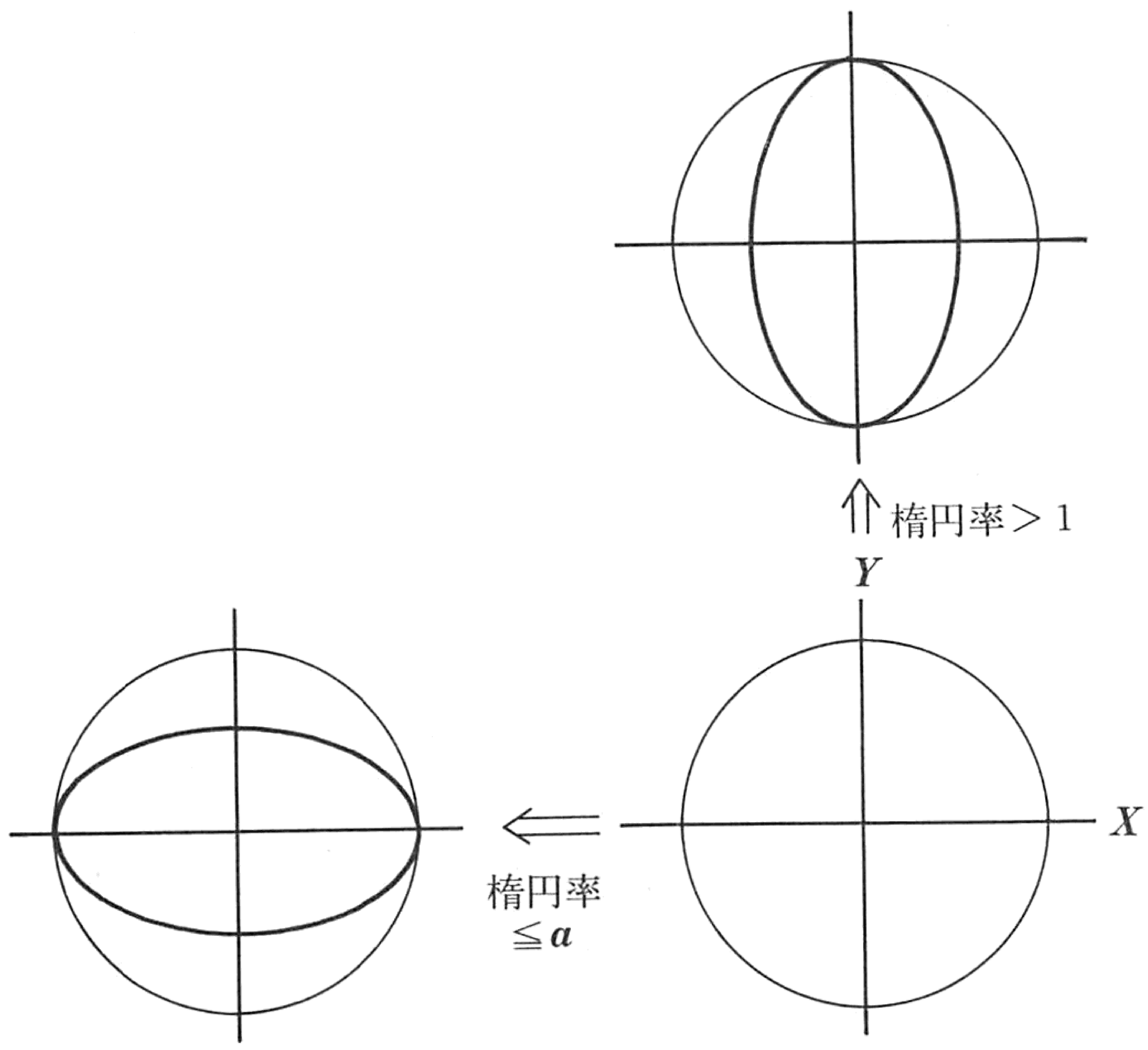


図7−2−12　楕円率

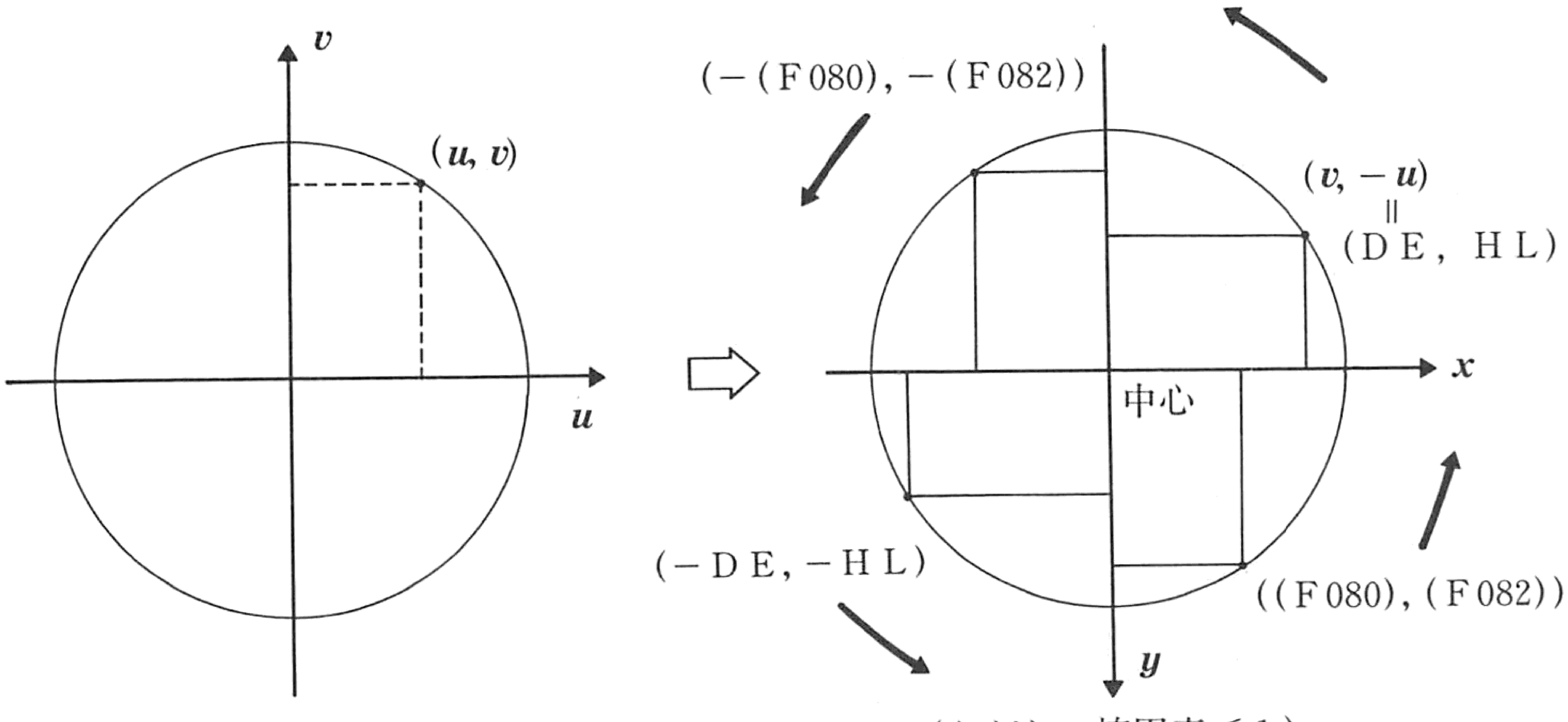


図7−2−13　$u-v$ 座標との対応

のとき楕円率も乗ぜられる)、それらの点に対して開始終了角度に対してどの位置にあるかを比較し、それに従って点をプロットしたり、中心から直線を引いたり、あるいは何も行わない、という処理をします。その後、(ｖ，ｕ）と（ｖ，ｕ）を回転した点についても行います。

角度のチェックはｕ－ｖ座標の円に対して行われ、(０，半径)から時計回りに数えたドット数（以後ドット位置という）のチェックに帰着されます。点（ｕ，ｖ）のｕ方向のドット位置は明らかにｕですから、この点を$\pi/2$，π，３$\pi/2$だけ回転させた点のドット位置は、八分円のドット数（Ｆ０７ＣＨ～Ｆ０７ＤＨに保存されていた）の２倍，４倍，６倍を加えたものになります。また、(ｖ，ｕ)のドット位置は八分円ドット数からｕを引いたものにさらに同様にオフセット（＝八分円のドット数）を加えたものとなり、回転された点のドット位置も同様に計算されます。

楕円率を乗ずる処理（７ＡＢ２Ｈ～７ＡＢＤＨ，サブルーチン：７ＢＡＢＨ～７ＢＤ６Ｈ）は、指定された楕円率が１より大きいか小さいかと、u-v 座標の点（ｕ，ｖ）の各座標値がＸ座標として使われる($\pi/2$，３$\pi/2$回転させたときはｕが、その他のときはｖがＸ座標として使われる）かＹ座標として使われるかによって楕円率を乗ずるかどうかが決まり、このサブルーチンではＸ座標として使われる場合の値をＤＥ、Ｙ座標として使われる場合の値をＨＬに返すようになっています。

それでは、８点プロットルーチンを詳細に見てみましょう。

まず、(ｕ，ｖ）のドット位置ｕをセット（Ｆ０８５Ｈ～Ｆ０８６Ｈ）し、ドット位置オフセット（Ｆ０７ＥＨ～Ｆ０７ＦＨ）をクリアします。そして、ｕ，ｖについてＸ，Ｙとして使われる場合の値を計算し、１回目のプロットに使われない方（次のプロットで使われ、ちょうど１回目にプロットする点の前の点に当たる）をストア（Ｘ：Ｆ０８２Ｈ～Ｆ０８３Ｈ，Ｙ：Ｆ０８０Ｈ～Ｆ０８１Ｈ）します。１回目の点はＤＥを反転して得ます。この後、４点（０，$\pi/2$，π，３$\pi/2$回転させた点）プロットをコールします。

４点プロットのルーチン（７Ｂ０２Ｈ～７ＢＡＡＨ）は、画面上の上右部分で指定された楕円上の点と、それに対応する上左，下左，下右部分の点をプロットするものです。これらの点はu-v座標では、第１象限内の指定された点と、$\pi/2$，π，３$\pi/2$だけ回転させた点に対応します。各点について、まずそのドット位置を計算します。始めに指定された点については、そのｕ座標がドット位置となっていますが、その他の点は、ドット位置オフセットを加えることで得られます。つまり、点を処理するごとに、オフセットを四分円のドット数だけ増加させます。こうして得られたドット位置と開始・終了角度との位置にしたがってプロット（７Ｂ４４Ｈ)、またフラグ（Ｆ０７ＢＨ）によって中心から直線を引く（７Ｂ３２Ｈ)、などの処理を行います。もちろん、ビューポート外の場合にはプロットは行いません。そして、次の点の中心相対座標を得てループします。次の点の座標は前回の点に対して中心に関し点対称な位置です。プロットするときには、中心の座標を加えて基本座標を計算しますが（７Ｂ８７Ｈ～７ＢＡＡＨ)、このときにｕ座標はＹ座標に加えられ、ｖ座標はＸ座標に加えられることに注意してください。

7−2−3　連結した領域の塗りつぶし(ペイント)

【1】　アルゴリズムの概略

連結した領域とはどのようなものか考えてみましょう。

境界と、境界上にない1点が与えられたとき（ペイント開始点)、その点を含む「連結した領域」を次のように帰納的に定義します。

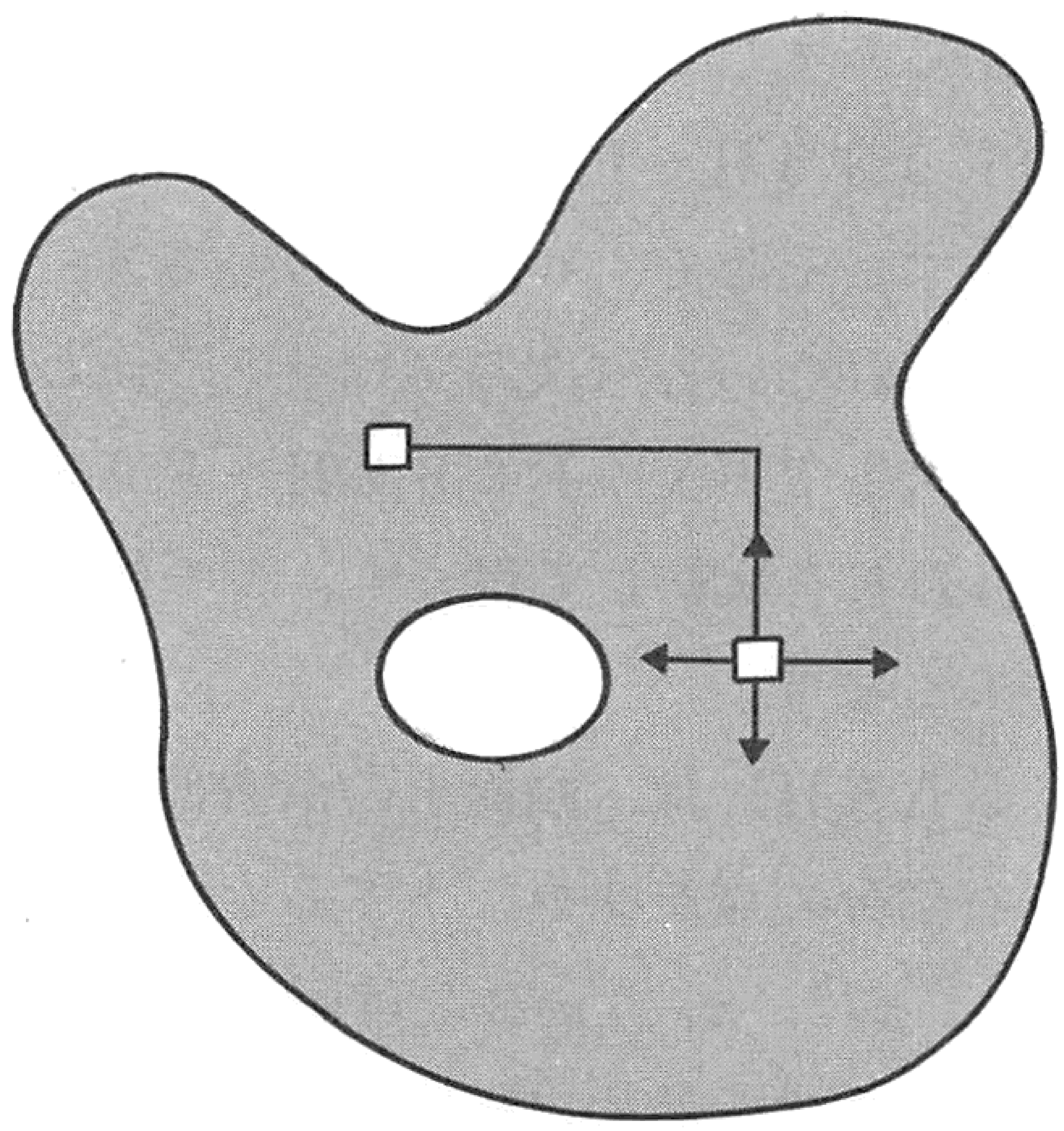
（1）　与えられた点は「連結した領域」の点である。

（2）　境界上でない点の上下左右少なくとも1つが「連結した領域」の点ならば、その点も「連結した領域」の点である。

（3）　（1）または（2）を満たすものだけが「連結した領域」の点である。

グラフィクス画面上なので、上下左右の意味は明らかでしょう。

したがって、この領域をペイントするにはこれを逆にたどっていけばよいことはすぐにわかります。つまり、指定された点から始めて、その上下左右についてそれが境界点であるかどうかを調べ、そうでなければペイントし、さらにその点の上下左右を調べる、というように行うのです。



□は1ドットを表す。
図中の2つのドットは
連結した領域に含まれる。

図7−2−14　連結した領域

このアルゴリズムは4点を同等に扱うものですが、現実のグラフィクス画面のドット並びをみると、横への移動が上下に比べて簡単に行える（ドット位置を表す1バイトを左または右へ1ビット回転させればよい）ので、上下よりも先に横方向のものを処理する方が効率が上がります。つまり（2）において左右へペイントできるだけ（境界点が現れるまで）ペイントし、そしてペイントした部分（このように連結しているものを横連結成分と呼ぶことにします。連結した領域から1ライン取り出したものとは異なることに注意）の上下についてさらに調査ベ

イントを行えばよいのです。もう少し検討してみると、上下のペイントもほとんどの場合、どちらか片方だけで済みます。それはたとえば上について考えると、上の横連結成分からはみ出していないときです。またそうでない場合でも上をペイントしてから、さらに下を調査ペイントするのですが、それが必要なのははみ出した部分の下に対してのみです。ところで、横連結成分はその左端の点と幅のドット数で表すことができます。これを用いてはみ出している部分の処理をペイントする方向というパラメータを加えると、次にペイントするべき方向と位置と幅を登録し、登録されたものに対してペイントするという方法ではみ出していないものと合わせて統一的に扱うことができます。はみ出していない部分はもとの部分をペイントしなくてよいからです。はみ出した部分に対する前のラインの処理は、左側にはみ出した部分の左端点とドット数、右側にはみ出した部分の左端点とドット数と、どちらも反対向きに登録します。もちろんはみ出した分も合わせていまペイントした横連結成分をペイントし、それを上下両方向について登録するように変更されます。

このような横連結成分を中心としたアルゴリズムでは方向は同じでも、いくつかの横連結成分に分割されて進む場合について考えなければなりません。分割された横連結成分それぞれについて登録しなければならないのです。

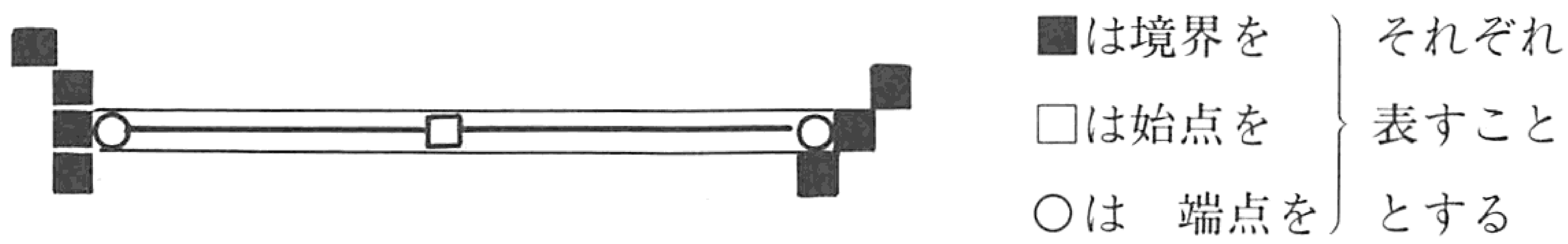
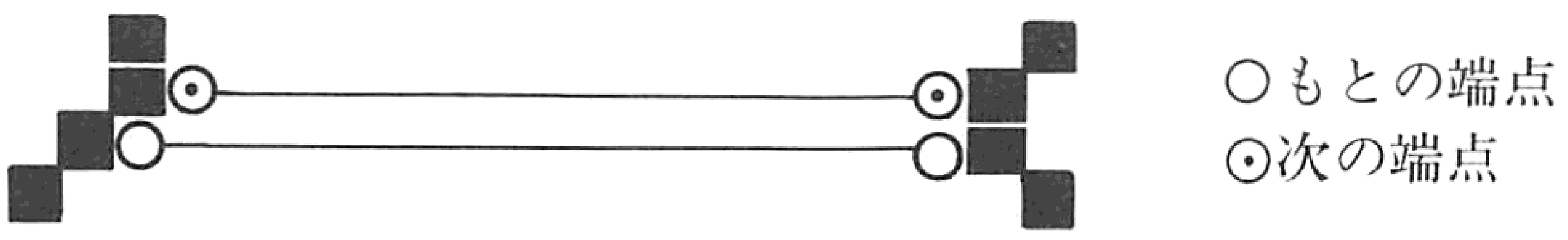
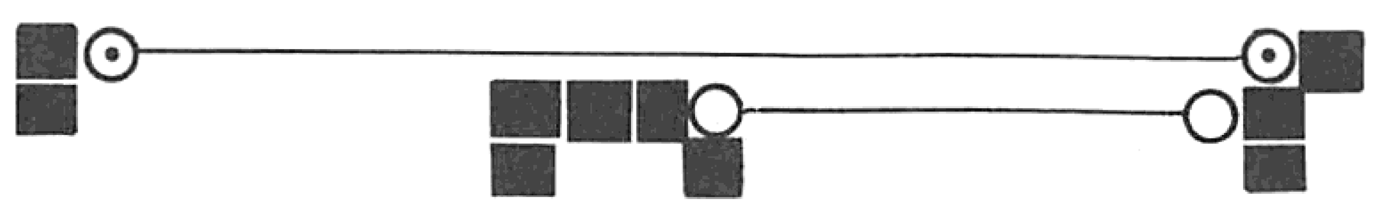


図7－2－15　横連結成分



a）はみ出していない場合



b）はみ出している場合

図7－2－16　次の横連結成分との関係



図７－２－17　方向づけられた横連結成分

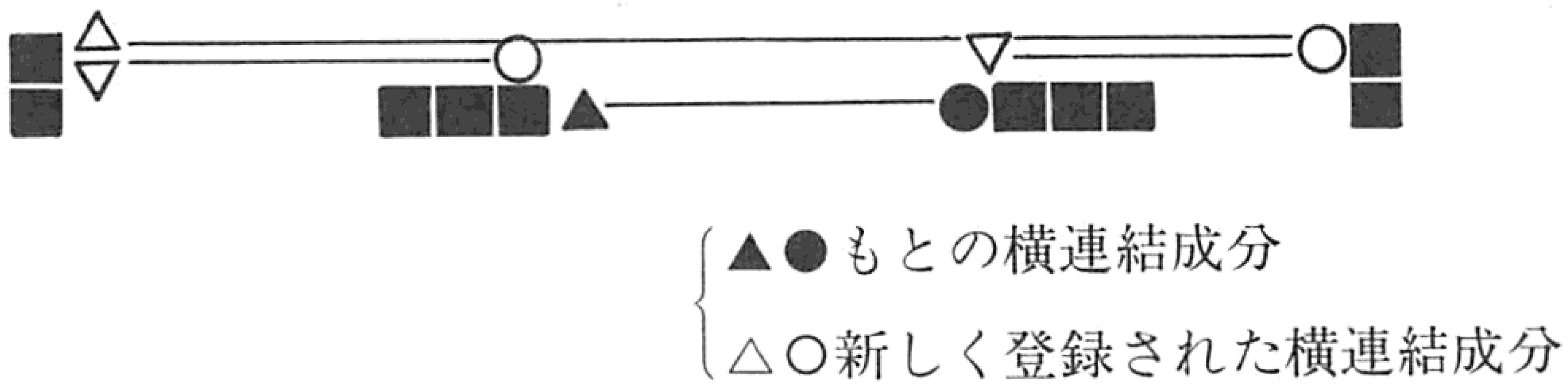


図７－２－18　はみ出した部分の処理

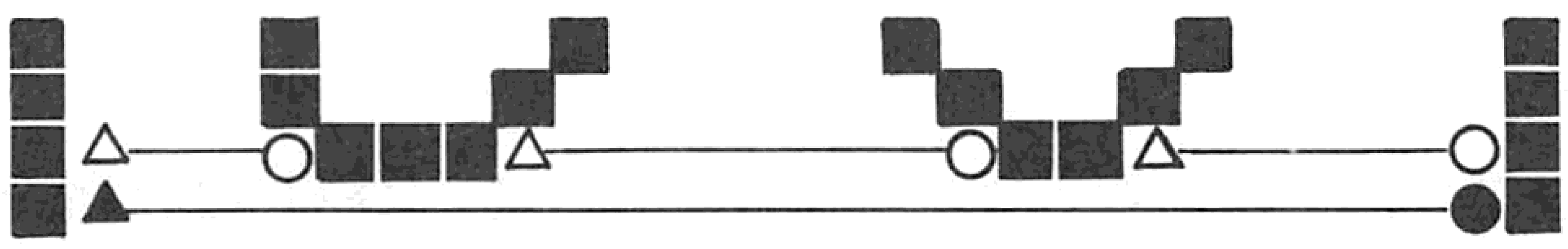


図７－２－19　いくつかの横連結成分に分割された１ライン

以上をまとめると、

［初期化］

① ペイント開始点を含む横連結成分をペイントし、その左端点を上下方向について登録

［点の取り出し］

② 登録されている横連結成分を１つ取り出す、登録されていなければ終了

③ 左端点をペイント方向に移動した点を計算、それに応じてビューポート左右限界パラメータ等をセット

［右側ペイント］

④ 右側をペイント、ペイントされなければ②へ

［左側ペイント］

⑤ 左側をペイント（左側にはみ出しているとき、これで両側がペイントされた）、左側がペイントされなかったら⑦へ

⑥ ペイントした部分（横連結成分の一部）を反対向きに登録

⑦ ペイントされた左端点、ペイントしたドット数(両側合わせて)、いまペイントしている向き、を登録

［分岐している領域のペイント］

⑧　前のラインにペイントしたドット数より右移動（このときにはペイントするとは限らない）したドット数が、少ないとき⑨へ、等しいとき②へ、多いとき（右に前よりはみ出したとき）⑭へ

⑨　前にペイントした幅の範囲で境界をスキップし（このときも右移動に数えられる）、次の横連結成分をペイントする

⑩　⑨においてペイントしたとき、境界をスキップしきらないうちに範囲を越えたならば②へ

⑪　⑨においてすでにペイントされていたならば⑧へ

⑫　⑨においてペイントした横連結成分を登録する

⑬　⑧へ

⑭　はみ出した部分を反対向きに登録する

⑮　②へ

ただし、すでにペイントされているところは移動するだけでペイントは行わないこととします。ペイントされなかった場合には、そこで、その位置・方向のペイントは打ち切られます。

【２】　実際のＰＡＩＮＴ文

上で示したアルゴリズムは実際のものと同じです。ただし、ＰＣ－８８０１のペイント機能にはタイルペインティングがあり、この処理はアルゴリズム中に現れていません。

この処理は実は非常に簡単で、ペイントの際、ペイントすべき部分にパターンを当てはめていけばよいのです。すなわち、サーチのときにペイントすべき部分がわかりますから、それに合わせてタイルパターンを切り出し、もとの状態と合成してストアするのです。ペイントされたかどうかの判定もパターン（の一部）と一致しているかどうかでなされます（したがって、逆に誤った判断を下すこともある）。また、タイルストリングとして、タイルパターンを複数行分指定することができますが、これは先頭のパターンを０ライン（基本座標で）とし、順に指定されています。

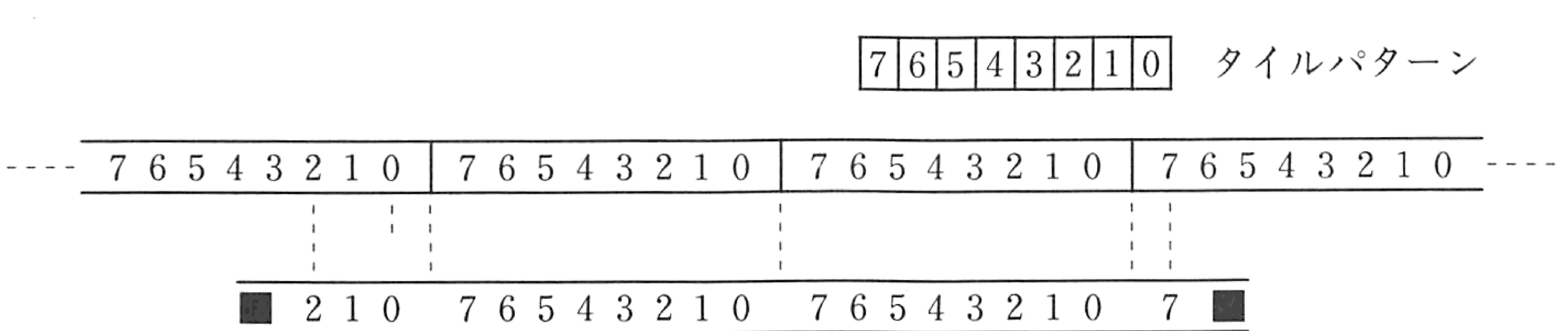


図７－２－20　タイルペインティング

横連結成分の登録には、テーブルの構造としていろいろなものが考えられますが、ここではキューが用いられています。これは、サーチポイントキュー（以下ＳＰＱと略記）と呼ばれ、登録するデータは点のプレーン・バイト数・ビット位置座標と向き＋タイルストリングカウンタ値とペイントしたドット数です。

PAINT文処理（７６７４Ｈ～７７ＤＤＨ）は多数のサブルーチンを持っていますが、大きくわけて、

（１）パラメータＧＥＴ＆チェック

（２）左右サーチ＆ペイント

（３）ＳＰＱ操作

に分けることができます。

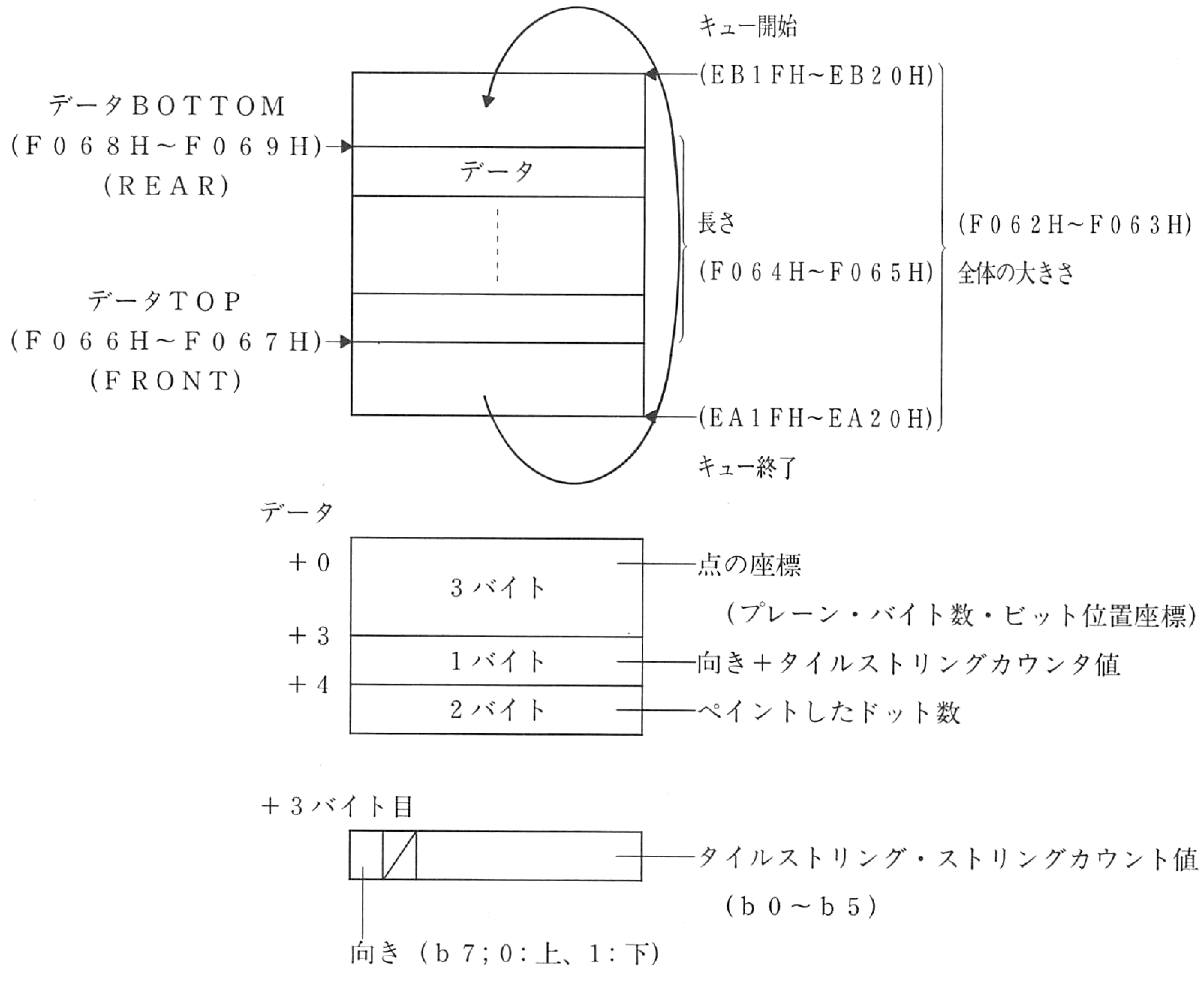


図７－２－21　サーチポイントキュー

（１）パラメータＧＥＴ＆チェックには、領域色タイルストリングＧＥＴ（７７Ｅ４Ｈ～７８４５Ｈ)、バックグラウンドストリングＧＥＴ（７８ＦＥＨ～７９３８Ｈ)、タイルストリングカウンタ初期化（７８９６Ｈ～７８ＡＥＨ)、タイルストリング読み出し設定（７８ＡＦＨ～７８ＤＢＨ)、タイルストリングエラーチェック（７８４６Ｈ～７８７ＡＨ)、などがあります。これらの多くは、テキストからパラメータを得てチェックするときに用いられます。

（２）左右サーチ＆ペイントには、左サーチ（７７ＣＤＨ～７７ＤＦＨ、本体は６２８ＢＨから始まる領域にある)、右サーチ（７７ＡＥＨ～７７ＢＦＨ、本体は６３１ＡＨからの領域にある）があります。特に重要なのが右サーチで、これは【１】の⑨を行うものです。

（３）ＳＰＱ操作には、ＳＰＱ初期化（７９３９Ｈ～７９７０Ｈ)、ＳＰＱに登録（７９７１Ｈ～７９９ＣＨ)、ＳＰＱより取り出す（７９９ＤＨ～７９Ｃ５Ｈ）などがあります。

次にメインルーチンと【１】で示したアルゴリズムの対応を示します。

７６７４～７６Ｂ３Ｈ	パラメータＧＥＴと初期化
７６Ｂ４～７６ＤＢＨ	①
７６ＤＣ～７６ＥＥＨ	②
７６ＥＦ～７７０ＦＨ	③
７７１０～７７１５Ｈ	④
７７１６～７７１ＤＨ	⑤
７７１Ｅ～７７２３Ｈ	１点だけ左にペイントしたときも反対向きにペイントすることはないので⑦へ
７７２４～７７３７Ｈ	⑥
７７３８～７７３ＦＨ	⑦
７７４０～７７４８Ｈ	⑨の一部
７７４９～７７５７Ｈ	⑧
７７５８～７７５ＣＨ	⑩
７７５Ｄ～７７５ＦＨ	⑪
７７６０～７７６ＥＨ	⑫
７７６Ｆ～７７７０Ｈ	⑬
７７７１～７７７９Ｈ	右に１ドット分多くペイントしただけならば反対向きにペイントする必要はないので②へ
７７７Ａ～７７８ＣＨ	⑭
７７８Ｄ～７７８ＥＨ	⑮

１ドット分はみ出した場合の処理が加えられています。１ドットはみ出した場合は、そのドットの上か下（ペイントしている向きと反対方向）が境界点であり、またはみ出した次のドットも境界点なので、反対向きにはペイントする必要がないのです。

バックグラウンドストリングの処理は、⑦の中（でコールしているサブルーチン中）でのみなされます。その内容は左右ペイント後、その横連結成分がすでにペイントされていれば、通常はそこはペイントが完了したとみなしてその成分は登録しないのですが、そのラインのタイルストリングがバックグラウンドストリングに一致していれば、点の登録をやめずに、強引に登録してしまうというものです。このパラメータはいかなる場合に用い、その結果どのような効果があるか定かではありません。ＰＣ－９８０１のペイントルーチンでは、このパラメータが無視されることがそれを証明しています。

最後にひとこと注意しておくと、ＳＰＱ初期化はPAINT文にエントリした直後に行われます。この後で種々のパラメータの評価、ＧＥＴが行われます。ＳＰＱ初期化処理では、フリーエリア（変数領域の後ろから文字列エリアの先頭まで）をフルに使用することができるようにように初期化されます（メモリの少ないときはガベージコレクションも行う）。したがって、PAINT文評価中では絶対にフリーエリアを消費するような式を用いてはいけないのです。最も簡単な例では、タイルストリングなどにＣＨＲ＄関数を用いてパラメータを指定することです。これにより文字列領域は拡大されますから、PAINT文実行後はこの部分は破壊されることがあります(複雑な図形をペイントしたときにはほぼ間違いなく破壊される)。上の状況は目で見ることができます。変数をたくさん使用した後、さきほどの例のようにタイルストリングをＣＨＲ＄の足し算で指定します。少し込み入った図形をペイントさせれば、途中からパターンが変わるのがわかります。

これ以上の詳しい情報は、「グラフィクス解析マニュアル」（尾高　著）とＲＯＭを参照してください。

INDEX

総集編

CONTENTS

PC－8801　INDEX総集編

★書籍名の略号

BASIC……PC－8801　N_{88}－BASIC解析マニュアル
ディスク……PC－8801　N_{88}－BASICディスク解析マニュアル
グラフィクス…PC－8801　N_{88}－BASICグラフィクス解析マニュアル
モニタ………PC－8801　N_{88}－BASICモニタ解析マニュアル
総集編………PC－8801　N_{88}－BASIC解析マニュアル総集編（本書）

（注）SR独自のアドレスは、ここには掲載されておりません。

【I】　メインROM（ROM3・4）

アドレス	書籍名	ページ	機　　能
１８Ｄ４Ｈ	総集編	３４	ＬＬＩＳＴエントリ
１８Ｄ９Ｈ	総集編	３４	ＬＩＳＴエントリ
１９４ＣＨ	総集編	３４	テキストを中間言語形式からＬＩＳＴ形式に変換する
１ＤＥ６Ｈ	ＢＡＳＩＣ	１８	単精度型実数の減算を行う
	総集編	１０７	
１ＤＥＡＨ	ＢＡＳＩＣ	２０	単精度型実数の加算を行う
	総集編	１０７	
１Ｆ１０Ｈ	ＢＡＳＩＣ	２２	単精度型実数の自然対数を求める
	総集編	１１９	
１Ｆ５３Ｈ	ＢＡＳＩＣ	２４	単精度型実数の乗算を行う
	総集編	１０７	
１ＦＢ７Ｈ	ＢＡＳＩＣ	２６	単精度型実数の除算を行う
	総集編	１０７	
２０ＡＢＨ	ＢＡＳＩＣ	２８	ＦＡＣの符号を反転する
	総集編	１０７	
２０Ｅ８Ｈ	ＢＡＳＩＣ	３０	単精度型実数の移動を行う
２１３４Ｈ	ＢＡＳＩＣ	３２	単精度型実数の比較を行う
	総集編	１０６	
２１５ＦＨ	ＢＡＳＩＣ	３４	整数の比較を行う
	総集編	１０５	
２１６ＥＨ	総集編	１０６	倍精度実数の比較を行う
２１Ａ０Ｈ	ＢＡＳＩＣ	３６	ＦＡＣのデータを整数に変換する
	総集編	１０９	
２１ＢＣＨ	ＢＡＳＩＣ	３８	単精度型実数を整数に変換する
	総集編	１０９	
２１ＦＤＨ	ＢＡＳＩＣ	４０	整数をＦＡＣに格納する
２２０２Ｈ	ＢＡＳＩＣ	４２	ＦＡＣの型を指定する
２２１４Ｈ	ＢＡＳＩＣ	４６	ＦＡＣのデータを単精度型実数に変換する
	総集編	１０９	
２２１ＣＨ	ＢＡＳＩＣ	４８	倍精度型実数を単精度型実数に変換する
	総集編	１１０	
２２２ＦＨ	ＢＡＳＩＣ	５０	整数を単精度型実数に変換する
	総集編	１０９	
２２３ＥＨ	ＢＡＳＩＣ	５２	ＦＡＣのデータを倍精度型実数に変換する
	総集編	１０９	
２２４６Ｈ	ＢＡＳＩＣ	５４	単精度型実数を倍精度型実数に変換する
	総集編	１０９	

アドレス	書籍名	ページ	機　　能
２Ｆ９１Ｈ	ＢＡＳＩＣ	１０４	単精度型実数の正弦（サイン）を求める
	総集編	１２４	
３０２ＣＨ	ＢＡＳＩＣ	１０６	正接値（タンジェント）を求める
	総集編	１２８	
３２０３Ｈ	総集編	１８６	カナ文字をＳＩ／ＳＯプロトコルでＣＯＭ１に送る
３２２６Ｈ	総集編	１８７	ＲＳ−２３２Ｃに１文字送る
３２４２Ｈ	総集編	１４８	キーボードマトリクスをスキャンし、キー状態を調べる
３５８３Ｈ	ＢＡＳＩＣ	１０８	キーボード１文字入力待ちを行う
	総集編	１４７	
３５Ｃ２Ｈ	ＢＡＳＩＣ	１０９	ＳＴＯＰキーのチェック
	総集編	１５８	
３５ＣＥＨ	ＢＡＳＩＣ	１１１	リアルタイムなキー入力を行う
３５Ｄ９Ｈ	総集編	１４７	キーボード用のキュー＃０を初期化する
３６９ＡＨ	ＢＡＳＩＣ	１１３	ディスクと１セクタの入出力を行う
	ディスク	４８	
	総集編	１９２	
３６ＤＢＨ	総集編	１９４	インテリジェントタイプのディスクの接続を調べる
３６Ｅ２Ｈ	ＢＡＳＩＣ	１１６	ＰＣ−８０３１のイニシャライズを行う
	ディスク	５１	
	総集編	１９３	
３７２２Ｈ	ディスク	５３	ＰＣ−８０３１にコマンド・パラメータを送信する
３７３ＡＨ	ディスク	５５	ＰＣ−８０３１からデータ列を読み込む
	総集編	１９６	
３７６６Ｈ	ディスク	５７	ＰＣ−８０３１へデータ列を送信する
	総集編	１９７	
３７９０Ｈ	ディスク	６０	コマンドの実行結果を受信する
３７Ｃ９Ｈ	ＢＡＳＩＣ	１１７	ＰＣ−８０３１へコマンドを送信する
	ディスク	６３	
	総集編	１９５	
３７Ｄ２Ｈ	ＢＡＳＩＣ	１１８	ＰＣ−８０３１へデータを送信する
	ディスク	６６	
	総集編	１９５	
３８４７Ｈ	ＢＡＳＩＣ	１２０	ＰＣ−８０３１からのデータ受信を行う
	ディスク	６９	
	総集編	１９６	
３９７５Ｈ	ディスク	７１	ＤＭＡ・ＦＤＤとのデータ転送を行う
３Ａ２ＤＨ	ディスク	７４	ＦＤＣのイニシャライズを行う

アドレス	書籍名	ページ	機　　能
4350H	BASIC	138	VRAM転送および属性設定を行う
	総集編	174	
4452H	BASIC	140	VRAM上のキャラクタを調べる
	総集編	174	
445BH	総集編	165	CRTCのアトリビュートコードを作成する
4472H	BASIC	142	スクリーン上のキャラクタを調べる
	総集編	176	
447DH	BASIC	144	カーソル移動および1文字表示を行う
	総集編	176	
45DDH	総集編	143	キューに1文字書き込む
45F8H	総集編	144	キューから1文字を得る
461BH	総集編	146	キューから得た文字を1つだけ戻す（連続使用可）
463DH	総集編	143	キューを初期化する
464EH	総集編	145	キューから得た文字を1つだけ戻す（連続使用不可）
4655H	総集編	144	キュー内に入っている文字数を数える
4667H	総集編	145	キューにあと何文字書き込めるかを数える
468CH	ディスク	115	ファイル・ディスクリプタの処理を行う
	総集編	140	
46F8H	ディスク	118	ファイルバッファのアドレスを求める
	総集編	136	
4742H	ディスク	119	ディスクの各パラメータをセットする
47F6H	ディスク	123	ファイルをオープンする
481DH	ディスク	125	ファイルのクローズ
4B1AH	BASIC	146	ホット・スタートを行う
4B54H	ディスク	126	シーケンシャルファイルへの出力
4B7BH	ディスク	128	シーケンシャルファイルからの入力
5059H	総集編	73	ON割り込みのチェックを行う
52BDH	総集編	62	NEXT文の処理を行う
5550H	ディスク	131	ライン・アウトプットを行う
559AH	総集編	98	ガベージコレクションを行う
5ACAH	総集編	100	変数のアドレスを求める、未登録の変数ならば登録する
5F92H	BASIC	150	スクリーン・エディットを行う
5FC8H	BASIC	148	ライン・インプットを行う
	ディスク	133	
	総集編	177	
6F06H	ディスク	134	ミニディスクユニットを両面仕様にセットする
6F0AH	ディスク	136	ドライブタイプの対応表を作成する

【２】　ＲＡＭ（主にディスクコード）

アドレス	書籍名	ページ	機　　能
８４００Ｈ	ディスク	２５	ジャンプテーブルのイニシャライズ
８４２７Ｈ	モニタ	１８３	＾Ｄコマンドエントリ
８４Ｆ０Ｈ	モニタ	１８９	＾Ｒコマンドエントリ
８４Ｆ１Ｈ	モニタ	１８９	＾Ｗコマンドエントリ
８５８８Ｈ	モニタ	１９５	ドライブ・サーフィス番号入力
８５ＢＡＨ	モニタ	１９８	論理トラック番号計算
８５Ｃ９Ｈ	モニタ	１９９	トラック・セクタ番号正当性チェック
８５ＤＢＨ	モニタ	２００	トラック・セクタ番号インクリメント
８５ＥＤＨ	モニタ	２０１	１セクタＲＥＡＤ
８５ＥＥＨ	モニタ	２０１	１セクタＷＲＩＴＥ
８５ＦＣＨ	ディスク	３３	ディスク入出力（８５ＥＤ，ＥＥＨのサブルーチン）
８６２３Ｈ	モニタ	２０２	ＨＥＬＰコマンド拡張ルーチン
８７９０Ｈ	ディスク	３４	ディスクＩＤセット
８７９６Ｈ	ディスク	３４	ディスクＩＤ・ドライブアドレスセット
８８Ａ４Ｈ	ディスク	１４０	グラフィック画面を指定データでクリアする
８８Ｂ９Ｈ	ディスク	１４２	グラフィック画面のスクロールを行う
８９Ｂ３Ｈ	ディスク	１４５	単精度型実数の逆正接値を求める
	総集編	１１３	
９２９ＢＨ	総集編	１１２	倍精度実数型の多項式の計算（奇関数）を行う
９２Ｂ０Ｈ	総集編	１１２	倍精度実数型の多項式の計算（一般形）を行う
９４８ＢＨ	ディスク	１４７	倍精度型実数の逆正接値を求める
	総集編	１１４	
９４Ｅ７Ｈ	ディスク	１４９	倍精度型実数の余弦値を求める
	総集編	１２７	
９４Ｆ０Ｈ	ディスク	１５１	倍精度型実数の指数関数の値を求める
	総集編	１１７	
９５６ＦＨ	ディスク	１５３	倍精度型実数の自然対数の値を求める
	総集編	１２０	
９５Ｃ３Ｈ	総集編	１２３	倍精度実数型のべき乗を求める
９６１５Ｈ	ディスク	１５５	倍精度型実数の正弦値を求める
	総集編	１２６	
９６５０Ｈ	ディスク	１５７	倍精度型実数の平方根を求める
	総集編	１２９	

アドレス	書籍名	ページ	機　　能
9691H	ディスク	159	倍精度型実数の正接値を求める
	総集編	128	
97EAH	ディスク	162	REMOVE処理を行う
9821H	ディスク	164	MOUNT処理を行う
9886H	ディスク	166	FATコピーのアドレスをセットする
98DAH	ディスク	167	クラスタ数からセクタ数への変換を行う
98EAH	ディスク	169	ディレクトリよりファイル名をサーチする
9953H	ディスク	172	ディスクのパラメータ・ドライブアドレスをセットする
9997H	ディスク	176	IDトラック・セクタをセットする
99A1H	ディスク	178	ディレクトリトラックをセットする
99EFH	ディスク	179	ネームバッファの内容を交換する
9BB6H	ディスク	182	ディスクよりファイルを削除する
9F69H	ディスク	184	ファイル名を出力する
A2F7H	ディスク	187	FATの読み書きを行う
A327H	ディスク	189	ディレクトリの読み出しを行う
A340H	ディスク	193	ディレクトリの書き込みを行う
A4FFH	ディスク	195	クラスタ・トラック変換
A53BH	ディスク	197	ディスクとの入出力を行う
A61DH	ディスク	201	ディスクの空きクラスタ数を求める
A679H	ディスク	203	ディスクファイルの大きさを求める
A6F1H	ディスク	205	IDセクタを読み出す
A74AH	ディスク	207	マシン語のセーブを行う
A7BBH	ディスク	209	マシン語のロードおよび実行を行う
A875H	総集編	64	WHILE文の処理を行う
A89FH	総集編	65	WEND文の処理を行う
AD48H	ディスク	212	プログラムのプロテクトのチェックを行う
AD5BH	ディスク	214	ライトプロテクトのチェックを行う
F155H	モニタ	251	ROM3・4内ルーチンコール
F16CH	モニタ	252	ROM1・2セレクト
F175H	モニタ	252	ROM3・4セレクト
F184H	モニタ	28	Mコマンド用LDIR
F187H	モニタ	28	Mコマンド用LDDR
F18AH	モニタ	252	1バイトリード
F1B5H	モニタ	255	1バイトライト

【3】　ＲＯＭ５（主にグラフィクス解析マニュアル）

アドレス	所属・備考	ページ	機　　能
６００ＤＨ	データ		ＲＯＭ５の処理アドレスのジャンプテーブル
６０４ＢＨ	ＣＩＲＣＬＥ	５４	画面の縦横比のパラメータを得る
６０５ＤＨ	GET＠, PUT＠	５５	ＧＥＴ＠，ＰＵＴ＠用ワークエリアの初期設定
６０ＥＡＨ	データ		ＰＵＴ条件サブルーチンのジャンプテーブル
６０Ｆ４Ｈ	ＧＥＴ＠	５７	１ドットラインのデータをＧＥＴする
６０Ｆ７Ｈ	ＰＵＴ＠	５９	１ドットラインのデータをＰＵＴする
６１１ＦＨ	ＧＥＴ＠	６１	１つのＧＶＲＡＭより１ドットラインＧＥＴする
６１６ＣＨ	ＧＥＴ＠	６３	ＧＶＲＡＭより１バイトＧＥＴする
６１８３Ｈ	ＰＵＴ＠	６４	１つのＧＶＲＡＭへ１ドットラインＰＵＴする
６１Ｆ０Ｈ	ＰＵＴ＠		カラーモード時、白黒グラフィックスパターンに色を付ける処理
６２１１Ｈ	ＰＵＴ＠		６１Ｆ０Ｈルーチンで使用するバックグラウンドカラーとフォアグラウンドカラーのマスタパターンの作成処理
６２２ＥＨ	ＰＵＴ＠	６６	ＧＶＲＡＭへ１バイトＰＵＴする
６２６０Ｈ	GET＠, PUT＠	６７	ＧＶＲＡＭのページ数を得る
６２６ＡＨ	ＰＡＩＮＴ		ペイントエリアのペイント（８ビット）とビューポート左端のチェックを行う
６２７ＢＨ	ＰＡＩＮＴ		ペイントエリアのペイント（８ビット）とビューポート右端のチェックを行う
６２８ＢＨ	ＰＡＩＮＴ		左端（左境界色またはビューポート左端）までのペイントのサブ処理
６２Ｃ３Ｈ	ＰＡＩＮＴ		ペイントエリア（８ビット分）のペイント
６２Ｅ９Ｈ	ＰＡＩＮＴ		領域色、境界色があるかどうかのチェック
６２ＦＦＨ	ＰＡＩＮＴ		マスクパターンチェック処理
６３１ＡＨ	ＰＡＩＮＴ		右端（右境界色またはビューポート右端）までのペイントのサブ処理
６３８３Ｈ	ＰＡＩＮＴ		最初の領域色、境界色の設定
６３９２Ｈ	ＰＡＩＮＴ	６８	境界色の設定を行う
６３Ａ６Ｈ		６９	ビューポート左端のチェックを行う
６３Ｂ６Ｈ		７０	ビューポート右端のチェックを行う
６３Ｃ８Ｈ		７１	ビューポート左右の境界アドレスを得る
６４３ＢＨ		７２	ＧＶＲＡＭのページ数を得る
６４４５Ｈ			ＷＲＩＴＥパターン用エリアへの転送ルーチン（ワークエリア←(ＨＬ～ＨＬ＋２))

アドレス	所属・備考	ページ	機　　能
6451H	PAINT	73	バックグラウンドストリングの省略値を生成する
646CH	PAINT	74	上のドットラインのパラメータをセットする
648EH	PAINT	76	下のドットラインのパラメータをセットする
64B0H			アクティブページのチェック
64C0H		78	ビューポートの範囲内外の判定処理　（その1）
64DAH		79	2つの数値の範囲内かどうかを判定する
6509H	CIRCLE	80	ワールド座標X値をスクリーン座標X値に変換する
6518H		81	スクリーンポインタをセットする
6574H	データ		マスクパターンテーブル
657CH		82	スクリーンポインタをインクリメントする
658EH		83	スクリーンポインタをデクリメントする
65A0H		84	直上の点のGVRAMアドレスを調べる
65C6H		86	直下の点のGVRAMアドレスを調べる
65F2H		87	レジスタにGVRAMの位置パラメータを得る
6606H		89	GVRAMの位置パラメータをセットする
661BH		90	指定する点の色を調べる
6638H		91	指定する点のON，OFFを調べる
6647H	LINE	92	指定した幅の1ドットラインを塗る
6686H	LINE	93	1バイトを塗る
66A0H		94	色の設定を行う
66B8H			カラーパレットNo.よりカラーマスクパターン獲得
66C5H			カラーパレットNo.とスクリーンモードよりアトリビュートを得る
66D7H		95	スクリーン上に点を描く
66EEH		96	指定するGVRAM上に点を描く
6700H		97	グラフィックス画面の初期化を行う
6734H		98	GVRAMのクリア
674AH		99	ビューポートを初期化する
6775H		100	グラフィックス処理用ワークエリアを初期化する
67A8H		102	ビューポート処理用ワークエリアを初期化する
681FH		104	ウィンドウを初期化する
6878H	メイン		COLORステートメント処理のエントリポイント
6882H	COLOR		COLORステートメント処理の開始アドレス
68ECH	COLOR		COLOR=ステートメント処理の開始アドレス
6916H	COLOR		中間言語（テキスト）から2つのパラメータを獲得（COLOR@処理のサブルーチン）
6927H	COLOR		COLOR@ステートメント処理の開始アドレス

アドレス	所属・備考	ページ	機　　能
６９８２Ｈ	ＣＯＬＯＲ		長方形の指定領域の、左上と右下の２頂点の座標を求める（右上と左下の２頂点で指定した場合）
６９８ＦＨ	メイン		ＳＣＲＥＥＮステートメント処理エントリポイント
６Ａ９４Ｈ	メイン	１０５	ＣＬＳ２ステートメント処理のエントリポイント
６ＡＣ６Ｈ	メイン		ＶＩＥＷステートメント処理のエントリポイント
６ＢＡ０Ｈ		１０６	ウィンドウとビューポートの横方向の比率を計算
６ＢＢＥＨ		１０７	ウィンドウとビューポートの縦方向の比率を計算
６ＢＤＣＨ	ＶＩＥＷ		中間言語（テキスト）より、ビューポートの座標（Ｖｘ，Ｖｙ）を得る処理
６Ｃ２ＢＨ	ＷＩＮＤＯＷ		中間言語（テキスト）より、ウィンドウの座標（Ｗｘ，Ｗｙ）を得る処理
６Ｃ５５Ｈ	メイン		ＷＩＮＤＯＷステートメント処理エントリポイント
６ＣＡ８Ｈ	メイン		ＶＩＥＷ関数処理のエントリポイント
６ＣＤ５Ｈ	メイン		ＷＩＮＤＯＷ関数処理のエントリポイント
６Ｄ０ＡＨ	メイン	１０８	ＬＰの初期化を行う
６Ｄ２９Ｈ	メイン		ＭＡＰ関数処理のエントリポイント
６Ｄ４８Ｈ	ＭＡＰ関数	１０９	ＭＡＰ関数の処理を行う
６Ｄ８ＢＨ	ＭＡＰ関数	１１１	ワールド座標からスクリーン座標への変換
６ＤＡ３Ｈ	ＭＡＰ関数	１１２	スクリーン座標からワールド座標への変換
６ＤＣ０Ｈ	メイン		ＰＯＩＮＴ関数処理のエントリポイント
６Ｅ２５Ｈ	メイン		ＰＯＩＮＴＯステートメント処理エントリポイント
６Ｅ２ＢＨ			中間言語（テキスト）から座標パラメータを獲得（その１）
６Ｅ３ＤＨ			中間言語（テキスト）から座標パラメータを獲得（その２）
６ＥＦ８Ｈ		１１３	ＬＰのワールド座標値をスクリーン座標値より獲得
６Ｆ３３Ｈ	ＬＩＮＥ		２つの数値を境界として、指定の数値の位置をチェックする
６Ｆ５ＦＨ	ＬＩＮＥ		ビューポート内に指定の点が存在するかどうかをチェックする
６Ｆ７ＡＨ	ＬＩＮＥ		ＬＩＮＥステートメント指定の２点(始点と終点)がビューポート内かどうかチェックする
６Ｆ９２Ｈ	ＬＩＮＥ		ラインの始点がビューポート内かどうかをチェックして、内部でなければクリッピング（ビューポートとの接点のアドレスを得る）処理を行う
６ＦＤ６Ｈ	ＬＩＮＥ		クリッピング処理の計算の一部
７０３０Ｈ	ＬＩＮＥ		クリッピング処理の計算の一部

アドレス	所属・備考	ページ	機　　　能
７０４２Ｈ			スクリーン座標Ｘを格納するエリア（Ｆ０１ＡＨ～Ｆ０１ＢＨ）とスクリーン座標Ｙを格納するエリア（Ｆ０１ＣＨ～Ｆ０１ＤＨ）の内容を交換する
７０５３Ｈ	メイン		ＣＯＰＹステートメント処理のエントリポイント
７０６４Ｈ	メイン	１１４	ＣＯＰＹキー処理（ＣＯＰＹステートメント処理）の処理開始アドレス
７１１ＣＨ	ＣＯＰＹ		プリンタへｎ／120インチの改行ピッチを指定する
７１４３Ｈ	ＣＯＰＹ	１１５	キャラクタ画面１行のプリント
７１６ＣＨ	ＣＯＰＹ	１１６	縦横変換用ワークエリアの設定
７１Ｂ１Ｈ	ＣＯＰＹ	１１８	縦横変換を行って１バイトのイメージデータを獲得
７１Ｃ１Ｈ	ＣＯＰＹ	１２０	グラフィックス画面１行のプリントを行う
７２０ＡＨ	ＣＯＰＹ	１２２	プリンタ８ビットドットモード設定コマンドの送出
７２２２Ｈ	ＣＯＰＹ	１２３	プリンタへ改行ピッチの設定コマンドを送る
７２２ＦＨ	データ		８ビットドットモードコマンドテーブル
７２３５Ｈ	データ		ｎ／120インチ改行ピッチ指定コマンドテーブル
７２３ＤＨ	ＰＵＴ＠	１２４	漢字フォントを読み出す
７２５８Ｈ	ＰＵＴ＠		漢字ステートメント処理ＰＵＴ＠処理開始アドレス
７２９０Ｈ	ＰＵＴ＠	１２５	フォントパターンのアドレスとワード数を得る
７２Ｄ０Ｈ	メイン		ＯＮ～ＧＯＳＵＢ処理のサブルーチン（その１）
７２Ｅ１Ｈ	メイン		ライトペン読み込み処理エントリポイント
７２ＥＤＨ	メイン		ＰＥＮ関数のエントリポイント
７３２４Ｈ	メイン		ＯＮ～ＧＯＳＵＢ処理のサブルーチン（その２）
７３ＤＦＨ	ＫＥＹ		ＫＥＹＬＩＳＴステートメントの処理開始アドレス
７４１ＡＨ	ＫＥＹ		ＫＥＹＬＩＳＴステートメントのサブルーチン（画面表示を行う）
７４２ＥＨ	メイン		ＫＥＹステートメント処理のエントリポイント
７４４３Ｈ	ＫＥＹ		ＫＥＹ定義処理の処理開始アドレス
７４８４Ｈ	ＫＥＹ		ＫＥＹＯＦＦステートメントの処理開始アドレス
７４８ＤＨ	ＫＥＹ		ＫＥＹＯＮステートメントの処理開始アドレス
７４９ＣＨ	ＫＥＹ		ＫＥＹＳＴＯＰステートメントの処理開始アドレス
７４Ａ１Ｈ	ＫＥＹ		ＫＥＹ（　）‥‥ステートメントの処理開始アドレス
７４ＥＥＨ	メイン		ＤＡＴＥ＄関数処理のエントリポイント

アドレス	所属・備考	ページ	機　　能
7514H	DATE$, TIME$		タイマバッファ内の時刻（または日付）データをキャラクタに変換する　（その1）
7526H	DATE$, TIME$		タイマバッファ内の時刻（または日付）データをキャラクタに変換する　（その2）
752AH	メイン		TIME$関数処理エントリポイント
754FH	メイン		ドライブ対応表作成処理エントリポイント
758AH	ドライブ対応表		ドライブ数のカウント
759AH	ドライブ対応表		ドライブ対応表の書き込み
75DDH	メイン		RENUMステートメント処理エントリポイント
7674H	メイン		PAINTステートメント処理エントリポイント
76B4H	PAINT	126	PAINTステートメントの処理を行う
7790H	PAINT		サーチポイントキュー登録処理
77AEH	PAINT	128	1ドットラインのPAINTを行う（右サーチ）
77C0H	PAINT	130	1ドットラインのPAINTを行う（左サーチ）レジスタ
77DEH			DEの内容を負の数値に変換する
77E4H	PAINT		中間言語（テキスト）より、領域色またはタイルストリングを得る
7846H	PAINT	131	タイルストリングのエラーチェックを行う
787BH	PAINT	132	タイルストリングエリアの終端をチェックする
7887H		133	バッファの比較を行う（DE～）−（HL～）
788FH			Aレジスタ × Bレジスタ の計算を行う(結果はレジスタBに得られる)
7896H	PAINT	134	タイルストリングカウンタの初期値を得る
78AFH	PAINT	135	タイルストリングの指定するパターンをセット
78DCH	PAINT	136	タイルストリングカウンタをデクリメントする
78EBH	PAINT	137	タイルストリングカウンタをインクリメントする
78FEH	PAINT		中間言語（テキスト）からバックグラウンドストリングを得る
7939H	PAINT	138	サーチポイントキューの初期化
7971H	PAINT	140	サーチポイントキューへ登録する
799DH	PAINT	142	サーチポイントキューより取り出す
79C6H	PAINT		フリーエリアのエンドチェックを行う
79D6H	メイン		CIRCLEステートメント処理エントリポイント
7AB2H	CIRCLE	144	楕円率を計算する
7ABEH	CIRCLE	145	円周上に8つの点を描く

アドレス	所属・備考	ページ	機　　能
7802H	CIRCLE		円周上の4点(X, Y)、(X, －Y)、(－X, Y)、(－X, －Y)を、指定するX、Yによりプロットする処理（7ABEHルーチンで2回使用)
7B87H	CIRCLE	147	現座標から円の中心に向かって線を引く
7B9CH	CIRCLE	148	円周上の猫画点の実座標を計算する
7BABH	CIRCLE		縦横比の処理
7BCAH	CIRCLE		レジスタDE×レジスタAの処理（結果はレジスタHLに得られる）
7BD7H	CIRCLE		中間言語（テキスト）から円弧の始点および終点を得る
7C28H	CIRCLE		FAC（単精度実数）を1と比較するサブルーチン
7C33H	メイン		GET@ステートメント処理、PUT@ステートメント処理のエントリポイント
7C42H	GET@		GET@ステートメント処理の処理開始アドレス
7CBDH	PUT@		PUT@ステートメント処理の処理開始アドレス
7D54H	GET@ PUT@		中間言語（テキスト）から配列のアドレスを得る処理
7D8AH	データ		PUT条件中間言語テーブル
7D8FH	メイン		ビューポートの範囲内外の判定処理　（その2）
7D98H			中間言語（テキスト）から、スクリーン座標系の座標パラメータを得る処理　（その1）
7DA7H			中間言語（テキスト）より、スクリーン座標系の座標パラメータを得る処理　（その2）
7DFBH	メイン		PRESETステートメント処理エントリアドレス
7E00H	メイン		PSETステートメント処理エントリアドレス
7E19H	メイン		POINT（Sx, Sy）ステートメント処理エントリアドレス
7E34H			中間言語(テキスト)からカラーパレットNo.を求める処理（その1）
7E37H			中間言語(テキスト)からカラーパレットNo.を求める処理（その2）
7E55H		149	スクリーン座標X方向の距離を求める
7E5FH			レジスタHLの内容を負の数にする処理
7E67H		150	スクリーン座標Y方向の距離を求める
7E72H		151	レジスタDEとF01C番地の内容を交換する
7E7CH			7E72H・727FHの両ルーチンを同時に行う処理
7E7FH		152	レジスタBCとF01A番地の内容を交換する
7E8BH	メイン		LINEステートメント処理エントリポイント

アドレス	所属・備考	ページ	機　　　　能
7EB7H	LINE	153	LINE‥‥，BFステートメントの処理開始アドレス
7EF2H	LINE		LINEステートメント処理の処理開始アドレス
7EF5H	LINE		LINEステートメント中核部分の処理開始アドレス
7F0BH	LINE		LINE‥‥，Bステートメントの処理開始アドレス
7F0EH	LINE	154	LINE‥‥，Bステートメントの中核部分の処理開始アドレス
7F38H	LINE		中間言語（テキスト）からラインスタイルを獲得
7F47H	LINE	155	直線を引く
7FBDH		156	レジスタDEを1/2にする
7FC5H	LINE		ラインスタイルに従って点を打つルーチン
7FD5H	LINE		中間言語（テキスト）から1バイトの整数値を獲得
7FEDH	COLOR		ディスプレイの種類（解像度等）をチェック

【4】 ROM2（6000H～7FFFH：モニタ解析マニュアル）

アドレス	所　属	ページ	機　　　　能
6135H	モニタ	36	Sコマンドエントリ
61A3H		41	Dコマンドエントリ1
61A9H		41	Dコマンドエントリ2
61E2H		44	ダンプメモリ本体
61EAH		44	ダンプメモリ本体（オフセット付き）
6254H		51	Xコマンドエントリ
6303H		66	レジスタネームテーブル
6330H		58	レジスタネーム・レジスタ内容表示
6395H		58	Fレジスタ入力
63C8H		67	フラグテーブル
63D8H		68	Fコマンドエントリ
6406H		70	Gコマンドエントリ
6479H		73	Iコマンドエントリ
6491H		74	Oコマンドエントリ
64A6H		75	Mコマンドエントリ
64D8H		78	Eコマンドエントリ

アドレス	所　属	ページ	機　　　　能
6562H	モニタ	82	数字処理
65BDH		86	1バイトスキップ
65C4H		86	1バイトバック
65CBH		87	カーソルアップ
65ECH		89	カーソルダウン
661BH		91	カーソルライト
662FH		91	カーソルレフト
6649H		94	ROLL UP
669CH		98	ROLL DOWN
66EEH		101	STOP・ESC処理
66F6H		102	カーソル移動
673DH		106	1画面メモリダンプ
6541H		109	サブコマンドテーブル
6754H		110	Wコマンドエントリ
67E9H		116	CMT1ブロック書き込み
6800H		116	CMT1バイト書き込み
6806H		118	Vコマンドエントリ
6807H		118	Rコマンドエントリ
68AEH		124	CMT1ブロック読み込み
68E4H		127	CMT1バイト読み込み
68F2H		128	ファイルネーム入力
694FH		132	USARTチェック
695EH		133	Bコマンドエントリ
6976H		134	Lコマンドエントリ1
697CH		134	Lコマンドエントリ2
69DCH		138	1行逆アセンブル
6ACBH		146	アセンブラ関係データテーブル
6D02H		149	Aコマンドエントリ
6D6BH		153	1行アセンブル
6E3AH		161	スキップスペース
6E46H		162	文字列比較
6E5FH		166	^Bコマンドエントリ
6E7AH		167	Pコマンドエントリ
6E8AH		208	プリンタ1行出力
6EE8H		213	文字列数値化1
6EE9H		213	文字列数値化2
6EEEH		213	数値入力（数値化3）

アドレス	所　属	ページ	機　　　能
6F1EH	モニタ	218	数値化メイン
6F5DH		222	1文字数値変換
6F80H		224	1バイト表示
6F9AH		226	アドレス表示
6FA7H		227	数値文字変換
6FCFH		229	1桁文字変換
6FD7H		230	2バイト比較
6FDDH		232	1文字入力
7025H		236	小大文字変換
702EH		238	文字列表示1
7037H		239	復帰改行
7044H		239	スペース表示
704CH		239	文字列表示2
7129H		243	1文字表示
7159H		245	キーボード入力待ち
7162H		245	リアルタイムキー入力
716EH		245	VRAMアドレス計算
717BH		246	カーソル表示
7183H		246	カーソル消去
718BH		247	CMT出力イニシャライズ
7194H		247	CMT1バイト出力
719DH		247	CMT出力クローズ
71A6H		248	CMT入力イニシャライズ
71AFH		248	CMT入力クローズ
71B8H		248	CMT1バイト入力
71C3H		249	ラインインプット
71D5H		249	BUSY入力
71DAH		249	プリンタ1バイト出力
727CH		168	TMコマンドエントリ
746FH		180	HELPコマンドエントリ
77E8H		107	EコマンドHELP

PC8801／mkⅡ／SR 解析マニュアル 総集編

1985年12月24日　初版発行
定価　2,800円

著　者　増淵文人　茜　照夫　島崎　孝

発行者　牧谷秀昭
発行所　秀和システムトレーディング株式会社
〒107　東京都港区南青山2－19－5
関原ビル
印刷所　東京スガキ印刷株式会社
ISBN4-87966-068-X C000 ¥2800E

秀和システム

秀和システムトレーディング株式会社　定価2,800円
ISBN4-87966-068-X C000 ¥2800E

秀和の解析技術、ここに結集！
いま、新たなる挑戦への序章(プロローグ)。

* ＊BASICシステム全体について、広く、深く、そして多角的に解説。
* ＊入出力・グラフィクス・数値演算など、応用性の高い話題を掲載。
* ＊既刊・解析マニュアルシリーズの活用に便利な、INDEX総集編。

● PC-8801/mkII/SR/TR/FR/MRに対応　解析マニュアル総集編

88 総集編

秀和システム

秀和システムトレーディング株式会社 〒107 東京都港区南青山2-19-5関原ビル 定価2,800円